■第１編　ソフトウェア基礎編■

第１章　ソフトウェアの感覚的体験

指導目標

ソフトウェアとはどのようなものであるかを実際にコンピュータ［ＰＣ、ＷＳ等］を使って極簡単なプログラムを自分で作り、その動作の具合を実感する。

ソフトウェアの入門は、“習うより慣れる”の世界である。概念から入るのではなく、実際に見て、触れて、やってみることから始めるのが肝心。

内容のあらまし

内　　容	目　　標	議　　論	机上実習	計算機実習
既存のプログラムを動かす	ソフトウェアの基本が、データの入力、計算、出力にあることを実感させる	↑ 特 に な し ↓	特になし	ＰＣ，ＷＳ コマンドをたたく
ファイルに関するごく初歩の知識	プログラムを作る上で必須となるファイルについて極初歩的な認識をさせる		特になし	ＰＣ，ＷＳ ファイルの基本コマンドをたたく
エディタについて	プログラムを作る上での補助作業として必須なエディタの動かし方を教える		簡単なテキストにつき編集をどうすべきか考える。	ＰＣ，ＷＳ ﾌﾟﾛｸﾞﾗﾑの編修用コマンドを実行

内　　容	目　標	議　論	机上実習	計算機実習
簡単なプログラムを作ってみる	データ入力→計算処理→結果出力の基本的な流れをプログラミング表現させる	特になし		ＰＣ，ＷＳ

ＰＣやＷＳをできれば一人に一台割り当て、実行確認できる環境を準備する。コンピュータの立ち上げ方および、コマンドの起動の仕方については若干説明する必要がある。ＰＣならＭＳ－ＤＯＳ、ＷＳならＵＮＩＸが揃っていることが望ましい。

以下の手順で実際にコンピュータを使って生徒にプログラムを実行させ、ソフトウェアの原理を実感し、把握させる。

１．既存プログラムを動かす

（１）標準入出力を伴う既存のプログラム（あるいはコマンド）を実行させる

①　対象プログラム

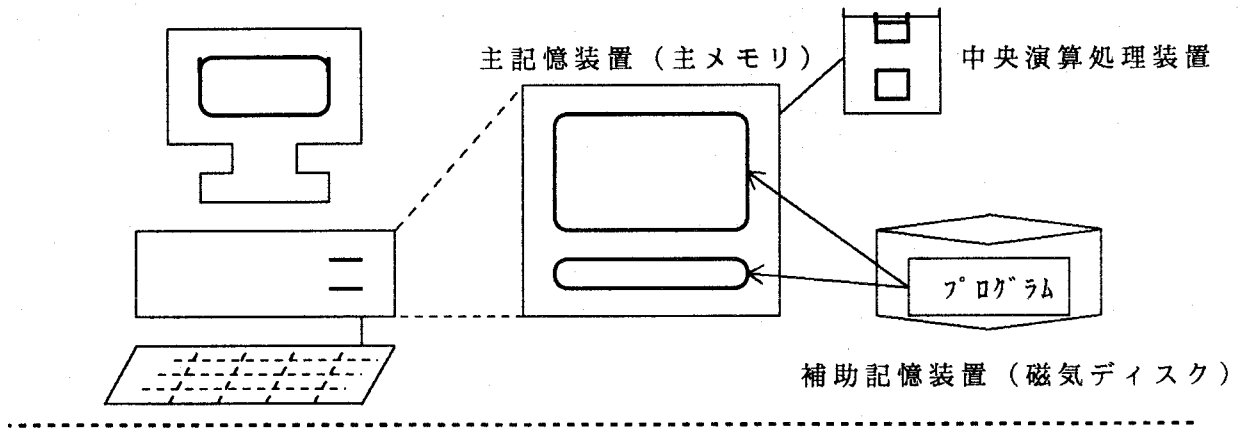
例えば、［合計値の計算、平均値の計算、簡単な対話処理］などの何種類かのプログラムを選ぶ。

プログラムは、ＢＡＳＩＣ，ＦＯＲＴＲＡＮ，ＰＬ／Ｉ，ＣＯＢＯＬ，Ｃなどで記述されていることが望ましい。対応するプログラムのソースコードを得ておく。

②　動作

コマンド起動後、ＫＢ入力をさせ、ディスプレー出力で結果を確認させる。

③　動作具合を図解する



④　ソースコードについて説明

実行動作の対象としたプログラム（あるいはコマンド）のソースコードを生徒に提示し、

・プログラムの概略

・処理の概略

－プログラムの開始の部分

－標準入力の部分

－処理の部分

－標準出力の部分

－プログラム終了の部分

について説明する。

（２）ファイル（テキストデータ）出力プログラムを実行させる

①　既存のテキストデータファイルの内容をプリント出力する

②　既存のテキストデータファイルの内容をディスプレー表示する

③　既存のテキストデータファイルを入力して、それに対し簡単な加工をし、他のファイルとして出力する。

□　データの例

・一連の文章データ

・電話番号帳データ　［所有者、電話番号、住所など］

・人事データ　［社員名、入社年月日、所属部門、キャリアなど］

・プログラムのソースコード

□　加工の例

・特定の言葉の抽出

・番号の若い順の並べ変え

・情報のつけ加え

・インデンテーションなどを付ける（いわゆるプログラムの清書）

関連知識　　文字コード

（３）ファイル（バイナリデータ）入出力プログラムを実行させる

①　既存のバイナリデータファイルの内容をプリント出力する（10進や16進）

②　既存のバイナリデータファイルの内容をディスプレー表示する(10進や16進)

---

③　既存のバイナリデータファイルを入力して、それに対し簡単な加工をし、他のファイルとして出力する。

□　データの例
- 文字列データをバイナリデータとみなす
- メモリの内容をダンプしたもの

□　加工の例
- １バイトずつ１６進表示する
- １バイトずつ８進表示をする
- １バイトずつ２進表示をする
- １バイトずつ取り出して、ビット位置を反転して表示をする

関連知識	２進表現　８進表現　１６進表現

２．ファイルに関する認識

プログラミングに必要な、最小限のファイルに関して認識すべき事項について学ばせる。ＯＳの違いにより若干コマンド名や指定書式などは異なるが、機能は基本的にはほとんど同様である。

（１）ディスクの初期化

ＦＯＲＭＡＴ　［ドライブ：］［ﾊﾟﾗﾒｰﾀ1］［ﾊﾟﾗﾒｰﾀ2］・・	－ＭＳ＿ＤＯＳ系

（２）ファイル名の付け方

形式　ｘｘｘｘｘｘｘｘ．ｚｚｚ

ファイルリストの表示

ＤＩＲ　［ファイル名］［ﾊﾟﾗﾒｰﾀ1］［ﾊﾟﾗﾒｰﾀ2］・・	－　ＭＳ＿ＤＯＳ系

ｌｓ　［ファイル名］［ﾊﾟﾗﾒｰﾀ1］［ﾊﾟﾗﾒｰﾀ2］・・	－　ＵＮＩＸ系

（３）ディレクトリの構造と作り方

ディレクトリの木構造について、ＭＳ＿ＤＯＳやＵＮＩＸのファイルシステムを例に図示して説明する。

ＭＫＤＩＲ　［ドライブ：］［パス名］	－ ディレクトリを作る
ＲＭＤＩＲ　［ドライブ：］［パス名］	－ ディレクトリを削除する

（４）ファイル操作コマンド　　（ＭＳ＿ＤＯＳ系のコマンド形式で説明）

ＣＨＤＩＲ　［ドライブ：］［パス名］	－ ディレクトリの変更、 カレントディレクトリの表示

ＦＣ　［ﾌｧｲﾙ名1］［ﾌｧｲﾙ名2］［ﾊﾟﾗﾒｰﾀ1］［ﾊﾟﾗﾒｰﾀ2］・・

－ ２つのファイルの内容を比較し、違いを表示

ＣＯＰＹ　［元ﾌｧｲﾙ名1］＋［元ﾌｧｲﾙ名2］＋・・［先ﾌｧｲﾙ名］

－ １つ以上のファイルを別の場所にコピー 、または連結する。テキストかバイナリファイルかの指定も可能。

ＤＥＬＥＴＥ　ﾌｧｲﾙ名［ﾊﾟﾗﾒｰﾀ］	－ 指定ファイルの削除を行う

ＤＵＭＰ　［ﾌｧｲﾙ名［開始ｱﾄﾞﾚｽ［最終ｱﾄﾞﾚｽ］］］［ﾊﾟﾗﾒｰﾀ］

－ ファイルの内容を１６進で表示する

（５）その他

ＤＡＴＥ　［ｙｙ－ｍｍ－ｄｄ］	－ 日付の表示や変更を行う
ＴＩＭＥ　［ｈｈ［：ｍｍ［：ｓｓ］］］	－ カレンダ時計の時刻表示と変更

3．エディタでプログラム・ファイルを作る

エディタを用いて編集し、プログラム・ファイルを作ること、また必要に応じその内容を修正することを学習させる。とりあえず、サンプル・プログラムを作成し、それを実行するところまでを目標とする。

エディタとしては、各情報処理技能者養成施設の標準のものを使用すればよいが、特にそれがなければ、“ＥＤＬＩＮ”、“ＳＥＤＩＴ”、“ｖｉ”コマンドなどが適当であろう。

編集および修正用サブコマンドとして、
・文字ストリングに関して、文字のつけ加え、変更、削除、挿入
・テキスト行に関して、コピー、削除、移動
等は必須である。

編集が終わったら、次にそのプログラムのコンパイルからランまでを実行させる。
以下の手順を踏ませる。

① ソースコードができたことを確認する
② コンパイルを実行しオブジェクト・ファイルができたことを確認する
③ リンクを実行しロードモジュールができたことを確認する
④ ランをする

上記②から④までの過程でエラーが発生したら、その原因をつかみプログラム・ファイルの編集ミスを指摘し、修正させ再度ランまでやり直させる。

4．簡単なプログラムを作り実行してみる

ＢＡＳＩＣ，ＦＯＲＴＲＡＮ，ＰＬ／Ｉ，ＣＯＢＯＬ，Ｃのいずれかの１種類以上を選択し、問題を提示し、それをプログラミング→エディット→コンパイル→リンク→ランのプロセスで実行させる。

ソフトウェアにおける処理には、
データの入力　→　データ処理用メモリ　→　データに対する計算処理　→
計算結果の求められる表現形式への変換　→　結果の出力　→　ファイルへの保存
という過程があり、それを通してコンピュータの入出力機構、コンピュータメモリの役割、演算機構などについてプログラムソースコードと実行動作の対応関係により理解させる。

学習の手順は以下の通り。

①　問題の提示　　数行から十行程度の文章で表せる問題を選択する
②　問題の理解　　何を求めているかを（入力、計算、出力の観点から）考えさせる
③　手続の作成（フローチャート）必要に応じてフローを書かせる
④　プログラミング　　問題に適合し易い言語でプログラミングさせる
⑤　コンパイル→リンク→ラン

どのような問題を提示するかは生徒のレベルなどを勘案の上指導者が決めることではあるが、ソフトウェアの入門中の入門ということもあり、以下の処理が含まれればよい。

①入力　　計算の基となるデータは、キーボード、ファイルから読ませる。文字データから２進データへの変換が必要であることも教える。
②出力　　結果の表現方法、２進データから文字データへの変換について学ばせる（極簡単なフォーマッティングを含む）
文字データの出力（タイトル、途中のコメントなど）も併せて。
③計算　　四則演算、繰り返し、判断を含める
④メモリ　　一次元配列を使う
⑤ファイル　　プリント出力でなく、結果をファイルに出力する。後にそのデータを入力するようなことを想定してもよい。

関連知識　　コンパイル、リンク、ラン

≪プログラム・サンプル１≫

$y = 2x^2 + 3X - 1$ という二次関数がある
Ｘの値（１～９）をキーボードから入力し、対応するＹの値を計算して画面に表示せよ。

但し、Ｘの値として９を入力したら、プログラムは終了するものとする。　Ｘの数値は整数で入れる。計算の回数は１０回以内とする。

ＦＯＲＴＲＡＮバージョンにて記述

```
nnnnnC                    ステートメンドフィールド                    ssssssss
      MAIN
C          COUNT = 最大の繰り返し数
C          X     = 入力されたＸ座標値
C          Y     = 計算された値
        INTEGER  COUNT, X, Y
C
      COUNT  = 10
C
      DO 10 I = 1,COUNT
        WRITE(6,1000)
        READ(5,100) X
        IF ( X .EQ. 9 ) GOTO 20
        Y = 2*X**2 + 3*X -1
        WRITE(6,2000) Y
   10 CONTINUE
C
   20 STOP
C
  100 FORMAT(I2)
C
 1000 FORMAT(' X no atai wo iretekudasai  ')
 2000 FORMAT(' Y no ataiha ',I3,' desu')
      END
```

応用として、上記の問題において、Ｘの値に１０以上の数値が入れられたらエラーメッセージを表示して、再度正しい値の入力を促すようなプログラムに機能を追加する問題を実施させる。

≪プログラム・サンプル２≫

３人の生徒の［身長（整数値３桁）、体重（整数値２桁）］をキーボードから入力する。

(i) まず、それらの値を入力して、画面に表示する。
(ii) 身長と体重の平均値を求め、それを画面に表示する。
(iii) 最高身長と最軽量体重を画面に表示する。

平均値も整数値でよい。

（１）ＦＯＲＴＲＡＮバージョン

```
nnnnnC                    ステートメントフィールド                    ssssssss
                                                                      SINTAI00
C         HIGHT =入力用身長変数、WEIGHT=入力用体重変数                SINTAI01
C         AVEHIG=平均身長用変数、AVEWEI=平均体重用変数                SINTAI02
C         MAXHIG=最高身長用変数、MINWEI=最軽体重用変数                SINTAI03
        INTEGER  HEIGHT,WEIGHT,AVEHIG,AVEWEI,MAXHIG,MINWEI            SINTAI05
C                                                                     SINTAI08
      AVEHIG = 0                                                      SINTAI09
      AVEWEI = 0                                                      SINTAI10
      MAXHIG = 0                                                      SINTAI14
      MINWEI = 100                                                    SINTAI15
C                                                                     SINTAI11
      DO 10 I = 1,3                                                   SINTAI12
        READ(5,100) HEIGHT,WEIGHT                                     SINTAI13
        WRITE(6,200) I,HEIGHT,WEIGHT                                  SINTAI16
        AVEHIG = AVEHIG + HEIGHT                                      SINTAI17
        AVEWEI = AVEWEI + WEIGHT                                      SINTAI18
C                                                                     SINTAI19
        IF ( MAXHIG .LT. HEIGHT )  MAXHIG = HEIGHT                    SINTAI20
        IF ( MINWEI .GT. WEIGHT ) MINWEI = WEIGHT                     SINTAI21
   10 CONTINUE                                                        SINTAI22
C                                                                     SINTAI23
      AVEHIG = AVEHIG / 3                                             SINTAI24
      AVEWEI = AVEWEI / 3                                             SINTAI25
      WRITE(6,1000) AVEHIG,AVEWEI                                     SINTAI26
      WRITE(6,2000) MAXHIG,MINWEI                                     SINTAI27
      STOP                                                            SINTAI28
C                                                                     SINTAI29
  100 FORMAT(I3,X2,I2)                                                SINTAI30
  200 FORMAT(' (',I1,') SINCHO = ',I3,', TAIJU = ',I2)                SINTAI31
C                                                                     SINTAI32
 1000 FORMAT(' HEIKIN SINCHO = ',I3,', HEIKIN TAIJU = ',I2)           SINTAI33
 2000 FORMAT(' SAIKO  SINCHO = ',I3,', SAISHO TAIJU = ',I2)           SINTAI34
      END                                                             SINTAI35
```

（２）ＰＬ／Ｉバージョン

```
SAMPLE:PROC OPTIONS(MAIN);

 DCL   EOF  BIT(1)  INIT('0'B);
 DCL   (HEIGHT,WEIGHT,A_HIGHT,A_WEIGHT,LONG,LIGHT)  DEC FIXED(9,2);

 ON ENDFILE(SYSIN)  EOF = '1'B;

    A_HIGHT,A_WEIGHT = 0;
    GET LIST(HEIGHT,WEIGHT);

    LONG = HEIGHT;  LIGHT = WEIGHT;  N = 0;

    DO WHILE (EOF = '0'B);
      N = N + 1;
      PUT SKIP LIST(HEIGHT,WEIGHT);
      A_HIGHT = A_HIGHT + HEIGHT;
      A_WEIGHT = A_WEIGHT + WEIGHT;

      IF LONG  < HEIGHT  THEN LONG  = HEIGHT;
      IF LIGHT > WEIGTH THEN LIGHT = WEIGHT;

      GET LIST (HEIGHT,WEIGHT)
    END;

    A_HIGHT   = A_HIGHT / N;
    A_WEIGHT = A_WEIGHT / N;

    PUT SKIP LIST ('AVERAGE  ');
    PUT SKIP LIST (A_HIGHT,A_WEIGHT);
    PUT SKIP LIST ('MAXIMUM HEIGHT & MINIMUM WEIGHT');
    PUT SKIP LIST (LONG,LIGHT);

END SAMPLE;
```

指導上の留意点

この章では、ソフトウェアとはどんなものかの感覚をを身につけさせることが目的であり、プログラミングやプログラムの規則についてあまり考えさせ過ぎないようにすべきである。

時間をとりすぎず、適宜ヒントを与え、全員がソフトウェアとは何であるかを感じとってもらうよう進行を工夫する必要がある。

コンピュータにスムーズには入れる生徒と、そうでない生徒とが当然出てこよう。お互いに分からない点を教え合うことにより、このレベルでは落ちこぼれが出ないよう十分気をつける必要がある。

第２章　ソフトウェアについての基本事項

指導目標

第１章で実感した内容を整理し、その深掘りを行う。できるだけ例題を通して原理を理解するようにする。

ソフトウェアの範囲は広い。現在の、複雑かつ大規模な情報ネットワーク環境はそれを統合的に理解するために、システムアーキテクチャー（体系）に基づきハードウェアやソフトウェアが組み込まれている。しかしこれを一度に教え込むことは大変であり、まず基本的な事柄から理解していくものとする。

本書では先ず、ソフトウェアとは狭義にはプログラムそのものを意味することを説明する。しかし広義には、要求仕様書や操作仕様書、運用マニュアル等も含まれることをも理解させる。

本書では、システム開発の全貌までは及ばないが、ソフトウェアの作成過程で必要となる基本的な技法については一通り教える。

内容のあらまし

内　　容	説　明	議　論	机上実習	計算機実習
ソフトウェアとは	ｿﾌﾄｳｪｱという概念について、一般のそれと情報技術においてとを対比で説明	ソフトウの概念を、広い見地から捉えるための議論をさせる	特になし	特になし

内　　容	説　明	議　論	机上実習	計算機実習
ソフトウェアの役割りについて	コンピュータ処理におけるｿﾌﾄｳｪｱの機能的な役割とその重要性について説明	最近の生活の中でソフトウェア的なものは何かを考えさせる	特になし	特になし
業務からソフトウェアへ	どのような対象、実務がｿﾌﾄｳｪｱ化されるか例を用いて説明する	世の中ではどんな業務がソフトウェアと関わりがあるか事例でで考える		特になし
ソフトウェアの作成手順	ｿﾌﾄｳｪｱが設計、開発、テストのﾌｪｰｽﾞﾄﾞ･ｱﾌﾟﾛｰﾁにより作られることを説明	フェーズドアプローチの必要性重要性について議論させる		特になし
ソフトウェアの動作手順	ﾌﾟﾛｸﾞﾗﾑのｺｰﾃﾞｨﾝｸﾞ完了後、翻訳、ﾘﾝｸ、実行、ﾃｽﾄ実ｼｽﾃﾑへの移行運用までの手順	テスト完了後実システムへの移行時に必要なことについて議論させる		汎用機 ＰＣ，ＷＳ
ソフトウェアの開発環境	環境として、人(開発要員)、物(ｺﾝﾋﾟｭｰﾀ、通信装置)、情報(ﾉｳﾊｳ、文書)を説明	・専門家 ・ﾊｰﾄﾞ、ｿﾌﾄ ・技術情報 の必要性につき議論させる		特になし

1．ソフトウェアとは

ソフトウェアという言葉は、ハードウェアとの対比により作られたものであるが、コンピュータ・プログラムとイコールであると理解してしまうと若干捉え方としては狭く、その存在の重要性に気づくことなく、狭量な技術者にとどまってしまう恐れもある。それゆえ、コンピュータに限定せず一般にソフトウェアという言葉が何を意味して使われているかについて事例を引用し説明する必要がある。

例えば、ＣＤにとっては楽曲がソフトウェアであり、ビデオテープレコーダーにとっては、ビデオショップに陳列されるビデオライブラリがソフトウェアである。放送用あるいは通信用衛星（ＢＳ，ＣＳなど）にとっては、衛星放送の番組がソフトウェアである。ＣＡＴＶにとってもしかりである。ハイビジョンＴＶもまた同様。レーザーディスク、ＣＤ－ＲＯＭ、光ディスク等にとっても、何をデータベース化してそれを活用するかが問題であり、そのデータベースとそれを利用するアプリケーションこそまさしくソフトウェアである。極端な話が、カメラにとってはフィルムは単なる情報伝達の媒体であり、被写された対象こそがソフトウェアである。

また、よりコンピュータに近いものとしては、ワープロ専用機、電子手帳あるいはファミコンの本体はハードウェアであり、ワードプロセッサとその利用者用マニュアルはソフトウェアである。時間管理ソフト、多国語翻訳ソフト、住所管理ソフトなどは電子手帳用ソフトウェアであり、ドラゴンクエスト、テトリス、信長の野望などはれっきとしたソフトウェアである。

これらのソフトウェアという言葉に共通する要素としては、ハードウェアを活用して人間の創造性を実現する元となるものだということである。これが重要である。

しかしながら上記のようなソフトウェアは比較的はっきりしており、少なくとも我々の五感で容易に認識できるものではある。それに反して、コンピュータにおけるソフトウェアはもはやそのような単純な時代を通り越し、情報処理環境の高度化、大規模化、複雑化にともない、相当量のプログラムがそれぞれの固有の機能を果たしながらも極めて微細な連携をとっている。しかもその大半は情報処理技術者にとっても気づかずにいながら大切な役を果たしているケースが多い。

コンピュータ、ソフトなければただの箱

コンピュータを活用する人々にとって耳の痛い言葉である。またソフトウェア技術者にとっては、身につまされる格言でもある。優れたソフトウェアを開発し続けなければ、たちまちその存在価値を疑われかねない。

我々が学習指導の対象とするソフトウェアは、企業における各種の職務に従事する者にとっての日常業務を確実に、効率的に行うための支援となるシステムであったり、公共団体における多様で迅速な住民サービスを可能としたり、また社会生活においては娯楽の対象となったり、人と人とのコミュニケーションを促進することに寄与する非常に重要なものである。

したがって、ソフトウェアとは、特定の機能をハードウェアに作動させるために必要な全ての情報を総称しており、情報の代表的な物件として下記を挙げることができる。

○　プログラム　－　コンピュータが理解できる一連の処理手続きやデータ群
○　仕様書類　－　プログラムを開発するに際して必要となる各種の文書で、プログラムの開発計画、プログラムに求められる要求事項、
プログラムの機能として実現して欲しい事項、プログラムを作成するための設計書、試験項目、保守に必要な事項を含む
○　使用説明書　－　利用者のために、プログラムが提供する機能について詳細に説明したもの。また、その円滑な運用のために、導入の方法や操作方法についても記述されている。

しかし概念的には、ハードウェア（コンピュータや各種ＯＡ機器も含み）からソフトウェア、さらには人間をも含む系全体が、どのような考え方（コンセプト）に基づいて、それぞれのコンピュータ資源が組み立てられ、またどのような役割を演じているかについても教える必要がある。それが広義のソフトウェアである。

このことを初等技術者（一般には第２種情報処理技術者）にソフトウェアとは何かの初期の段階から理解させることは容易ではないが、９０年代に入りますます複雑化するコンピュータ環境におけるソフトウェアの重要性を感じてもらうためには避けて通れない部分である。例えば、いわゆるメインフレームメーカの提唱するシステム統合アーキテクチャーのような、あるコンセプトに基づいたシステムの構成方法や、そこにおけるソフトウェアアーキテクチャー、個々のソフトウェアの役割と重要性とを説明するのも一つの方法である。メーカのパンフレットを使うことも好ましい。

また、どのようなプログラムが存在するかを知ってもらうために、代表的なソフトウェアのパンフレットなどを生徒に見せることも必要であろう。できれば、ソフトウェアショウやデータショウなどの見学を通じてプログラムの働きや意義を学んでもらうことも重要である。

2．ソフトウェアの役割について

前節［ソフトウェアについて］を補足すると、ソフトウェアとは、ハードウェアの持つ計算（演算）能力、情報（データ）の保管機能、コンピュータ機器間の情報（データ）の送受信能力を活用して、あらゆる産業界の企業、政治・行政・社会・文化などの公的な機関、学校、さらには家庭生活においてさえも対象となる、各種の仕事の処理を確実に、効率的に進行させる実体であると定義できる。

すなわち、コンピュータや通信機器に対して、色々な業務をどのような単位で、どのようなルールに基づき、どのような順序で進めるべきかを予め教え込み、ある号令や指示に従って動作を開始し、処理を進める理論の集合である。

ソフトウェアの役割については、ハードウェアとの機能的な役割分担から論じたりコンピュータ環境におけるソフトウェアの役割を中心にみたり、また、ソフトウェアがどのように産業や社会生活に役に立っているかを考えるなど多面的に論じることもできる。

（１）　ハードウェアとの機能分担での見方

コンピュータに搭載されるソフトウェアの量が増えるとともに、単位時間当たりの処理速度を上げるためには、全てをソフトウェアで解決しようという考えには限界がある。このため、極めて頻繁に使われまた高速性を保たねばならない処理部分について、その処理をハードウェア化する工夫がなされている。
その代表的な例として、

・ＯＳのロジック（例：タスクスイッチングメカニズム）の命令化
・浮動小数点演算をソフトウェアライブラリから、演算機構化する
・仮想メモリ制御をファームウェア化する
・専用マシンで処理する（例：データベースバックエンドマシン、通信用フロントエンドマシン、画像処理専用マシンなど）

が挙げられ、ソフトウェアとしての役割ではなくなっている。

（２）　コンピュータ環境におけるソフトウェア部分

ソフトウェアの中核はプログラムであると思ってよいが、これからのコンピュータは、コンピュータ、通信ネットワーク、各種ＯＡ機器などの複合体として捉える必要がある。つまりこれからのコンピュータは単なる単体としてのハードウェアとしてのみ捉えるのでなく、ネットワークの中に個々に役割を持ったコンピュータが散在し、互いに情報を交換し合いながら連携をとっている環境であるとみなすべきである。（これを最近はネットワークコンピューティングと呼んでいる）

したがって、ソフトウェアは単にそれが搭載されるハードウェアを動かすプログラムという観点でなく、また、コンピュータの利用者が直接それぞれのハードウェアに接触するかどうかは別として、コンピュータ環境に存在するあらゆる資源を効果的に活用するために各種ソフトウェアが介在し、それらが連携をとることにより系全体が旨く作用していることを認識させる必要がある。

ハードウェア	システム	
グローバル ネットワーク	WAN，VAN 専用回線、公衆回線、DDX－P、ISDN オンラインアプリケーション 通信制御用フロントエンドマシン（基本通信ソフト）	広域ネットワーク
基幹業務用 コンピュータ	大型汎用コンピュータ　基幹勘定系データベース 大型汎用コンピュータ　情報系データベース 　勘定系、情報系アプリケーション 　信頼性、安全性、効率性確保ソフトウェア スーパーコンピュータ　大型計算ソフトウェア OLTP（オンライントランザクション処理専用）	構内ネットワーク群
ローカル ネットワーク	ゲートウェイサーバーマシン（プロトコル変換） PBX（プロトコル、データフォーマット変換） ルータ、ブリッジ、リピータ	
業務部門用 コンピュータ	スーパーミニコンピュータ　部門データベース ハイエンドワークステーション　部門データベース 　業務部門専用計算処理ソフトウェア 　クライアントコンピュータの管理制御	
業務部門用 PC，WS	パソコン　個人情報管理、個人用情報処理環境 ﾜｰｸｽﾃｰｼｮﾝ　個人情報管理、個人用情報処理環境 　個人データベース 　業務用ソフトウェア MML　ネットワークソフトウェア	部門ネットワーク
OA機器	構内回線（LAN）　FAX　電話　ワープロ PC　イメージリーダ　表計算ソフト　統計ソフト スプレッドシート	

マイクロ・メインフレーム・リンク　部門間情報共有

3．業務からソフトウェアへ

情報処理技能者養成施設（コンピュータ・カレッジ）を卒業後の進路はいろいろであろうが、情報産業界、あるいは一般企業における情報処理部門、事務計算部門、エンジニアリング業務部門などの職務につくことも大いに予想される。

情報処理に強く関連する業務分野を大別すると以下のようになる。

（１）事務処理分野業務

企業における基幹業務系の勘定系システム（オンライン処理システム形態をとることが多い）、情報系システム（オフラインバッチ処理システム形態をとることが多い）での計算業務において、ソフトウェア開発が行われる。

ＣＯＢＯＬ、第四世代言語等を用いての業務システム開発（特にトランザクション処理システム）がその典型である。

また、情報系においては、各種統計処理、需要予測、経済分析など長期的な経営見通しに資する計算ソフトウェアを開発もしくは利用する場合が多い。

（２）科学技術計算分野業務

エンジニアリング業務部門における、各種数値計算、シミュレーション、工学系データ処理においてソフトウェア開発やパッケージソフトウェアの利用が行われる。

構造物、流体、熱、振動問題など多量の技術計算と、系の振る舞いを画像で解析したり、挙動をリアルタイム表示する業務が多い。

（３）マイクロエレクトロニクス関連分野業務

マイコン制御系のソフトウェア開発需要は非常に高い。各種家電製品、産業用機械、工場における生産ラインの制御などソフトウェア開発の機会が極めて多い。

（４）コンピュータサイドシステム運用分野業務

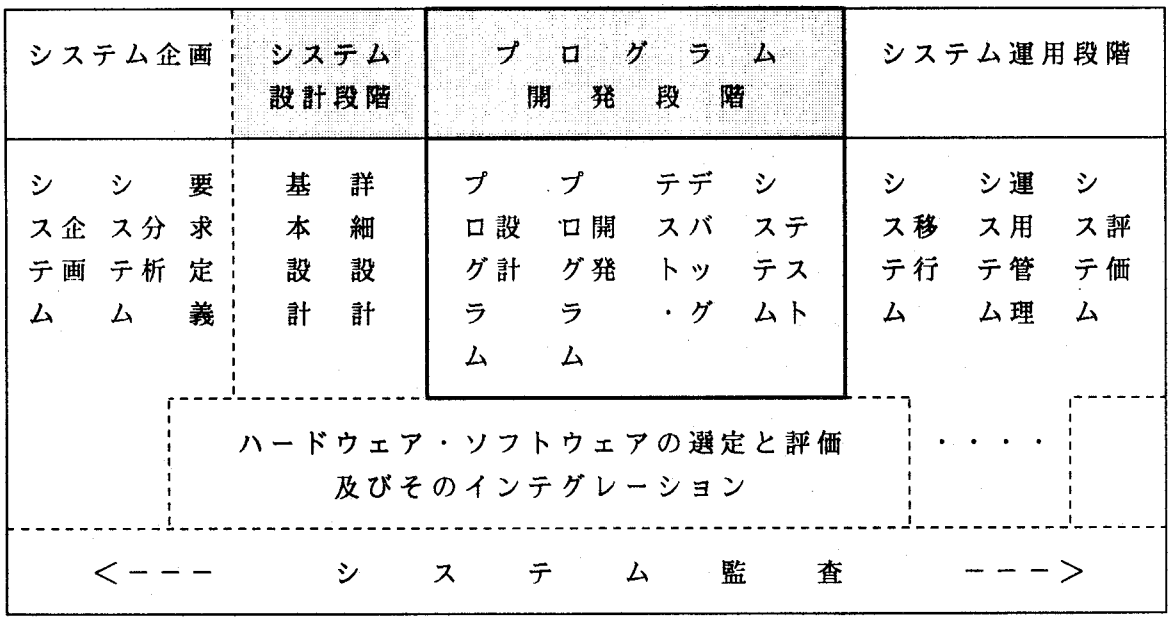
オペレーティングシステム・制御プログラムのジェネレーション、インストール、カスタマイズ、保守、コンピュータ利用部門への技術サポート、一般ソフトウェア開発等最も専門的な立場でソフトウェアに関わりを持つ。

4．ソフトウェアの作成手順

情報処理技術に関連する業界用語としては、ソフトウェアという概念は本来非常に広きに亘るものであるが、ソフトウェアの開発あるいは作成という場合、たいていはプログラム作成（開発）を意味する。したがって以後本書では特に断りをしない限り、ソフトウェア作成（開発）とプログラム作成（開発）とを同意語として扱う。

しかし、作成されるソフトウェアの機能範囲、処理の対象とする範囲が大きくなるほど、単純にソフトウェアあるいはプログラムの作成というときは、ソフトウェア開発プロジェクト全体の中のある限定された局面における作業を指す場合が多い。そのようなプロジェクトを通常システム開発という。

したがって、ソフトウェアの作成という局面はシステム開発全体においてどのような部分であるかについて予め知っておく必要がある。この概要については必ず教えなければならない。ここでいうシステムとは、電算化される大きな業務システムのことであり、それは複数のサブシステムから構成される。さらに各サブシステムはいくつかのプログラムに分割されて設計されることになる。



上図において、プログラミング経験の少ない生徒に当該用語を用いて説明することは適切ではなかろう。例えば、建屋を建築する場合を例にたとえを用いる方法が望ましい。

なお、第２種クラスの初等情報処理技術者にとって、というよりも専門学校、情報処理技能者養成施設の生徒のような基礎学力習得中の技術者にとっては、卒論、もしくは委託の特定プロジェクトに係わる以外では、システム開発といえるほどの大きなソフトウェア開発には出会わない。

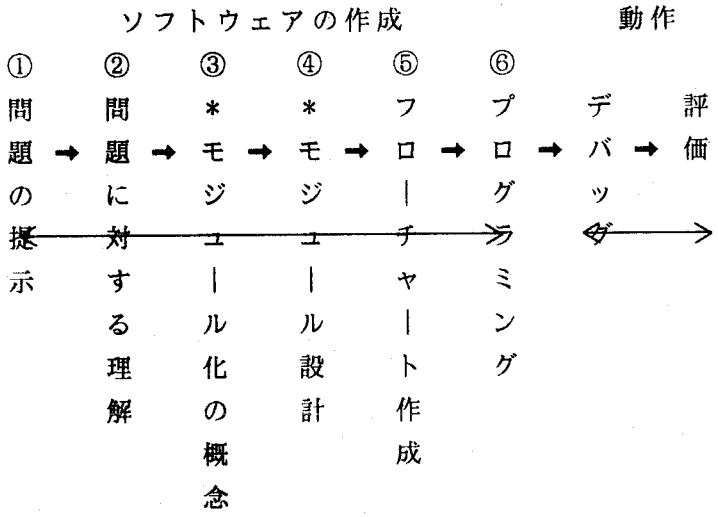
その意味で、厳密にはソフトウェアの作成局面では上図のシステム設計段階、プログラム開発段階を通して行う作業が発生するものの、本“ソフトウェアの基礎”という学習の範囲では、実際にはもっと省略されたステップによる開発手順を示すことで充分である。

本編では、ソフトウェアの作成手順を説明する上で、提示する問題文については、高々１頁をフルに使えば記述できる程度のものを扱うこととする。また、この段階でプログラミング技法として学習する内容としては、

・数値としての四則演算、大小比較
・論理演算
・文字としての比較
・条件判定
・簡単な入出力書式
・数値ライブラリ関数
・メモリ［一次元配列］
・繰り返し
・サブルーチン（ネストはしない）
・モジュール化の概念

に限定する。２次元配列や構造体などは含めない。

ソフトウェアの作成手順は、以下のプロセスを標準とする。



この場合、いきなりプログラムを書かせることは現実的でなく、まず学習すべき内容を含んだサンプルプログラムをいくつか提示し、プログラミングのやり方について強制的に覚えさせる。それを例示に処理内容を解説する。

また、授業の進め方としては、生徒全員に一様に解かせるだけでなく、ある生徒の模範例や、問題を含む解答について、公表し、全員で議論し合うことが生徒の思考力を強化する方法である。

生徒が、提示された問題を見てそれをプログラミングするまでにいたる過程をソフトウェアの作成（前頁）と設定する。この場合、③、④のモジュールの考え方については、いきなり持ち出すことは適当でなく、①→②→⑤→⑥から指導を始め、メインプログラムの範囲で一応のプログラムが書けるようになった段階でモジュール化の考えを導入する。

問題の候補として、
- ○　最小値、最大値、平均値の算出
- ○　正の数、負の数の書きだし
- ○　商品ファイルの明細を印刷
- ○　データのチェック（数値か否か、限界値内か、昇順か）
- ○　２次方程式の解
- ○　ベクトルの内積

等が適当であろう。

プログラミングの初心者に対しては、まずお手本を見せ、そこで行われている処理を絶対的なものと信じ込ませてやらせてみることが肝要である。とにかくひたすらものまねの時期である。

お手本を見て、コーディングシートに筆記させるか、エディタを使って入力させるかは授業環境等により決められるであろうが、できるだけ端末に座ることが望ましい。コマンドを覚えたり、エディタになれるいい機会である。どの言語を採用するかは、指導教官の流儀あるいはコンピュータ・カレッジの方針から決まるであろう。第２種情報処理技術者試験に使われる［ＦＯＲＴＲＡＮ，ＰＬ／Ｉ，ＣＯＢＯＬ，Ｃ］等が推奨されるが、マシン環境がＰＣやＷＳである場合、ＦＯＲＴＲＡＮ，Ｃのいずれかが得策であろう。

5．ソフトウェアの動作手順

ソフトウェアの開発フェーズ、実システムへの移行フェーズ、実際の運用フェーズにおいて、どのようなソフトウェアが、どのような手順で動作するかを説明する。

（１）ソフトウェア開発フェーズ

開発局面では、複数の技術者（設計者、プログラマ、データ入力員など）により、複数個のプログラムが別々に作られる。それぞれは全く独立のプログラムであったり、互いに依存関係を持ったり、主従関係にあったりする。したがって、それらが大きな１つのシステムへと統合されるときに、全体的な動きの中で個々のプログラムが真に正しい動作をしているか、またプログラム間での情報連絡はうまく進められているか確認されなければならない。

個々のプログラムの動作手順として、以下の手続について説明する。

(a) プログラム・ファイルの作成

プログラミングされた原始コードはエディタなどによりコンピュータ端末より入力され、コンピュータ周辺装置内に蓄えられる。これをコンパイル、リンクを行うとバイナリコードに変換される。

また、プログラムの実行時に入力情報として与えられるデータも原始プログラム同様、エディタなどによりコンピュータ端末より入力され、コンピュータ周辺装置内に格納される。

① 原始プログラムファイル（呼び名：ソース・コード・ファイル等）
② 目的プログラムファイル（呼び名：オブジェクト・コード・ファイル等）
③ 実行プログラムファイル（呼び名：エグゼキューション・ファイル等）
④ データファイル（呼び名：ソース・データ・ファイル等）

(b) 原始プログラムの翻訳

ソース・コードの記述言語（プログラム専門技術者の解釈可能な形式、コンピュータからみると文字情報としてのみ判別可能な形式）の違いにより、どの言語の翻訳プログラムが使われるかが選択される。翻訳プログラムが、原始プログラムファイルを入力し、翻訳後オブジェクト・コードに変換し目的プログラムファイルを作り出す。

① アセンブラ（機械語翻訳・プログラム）
② 高水準言語翻訳プログラム（ＦＯＲＴＲＡＮ、ＣＯＢＯＬ、Ｃ、ＰＬ／Ｉ等のコンパイラ・プログラム）

(C) 実行プログラムの作成

目的プログラムは、コンピュータ上での完全な実行形式にはなっていない。入出力ライブラリとの結合、サブプログラムとの結合をリンカ（連携、編集プログラム）により行って実行形式プログラムに変換される。

但し、この時点ではモジュール結合の概念を教える必要はない。

(d) プログラムの実行（テスト）

実行形式プログラムを、ローダによりコンピュータ上に読み込み、組んだプログラムの処理論理が正しいかどうか確認する。

原始プログラムの翻訳から実行形式プログラムのテストまでの作業は、それをどのようなコンピュータ環境で行うかにより、若干作業の進め方が異なる。

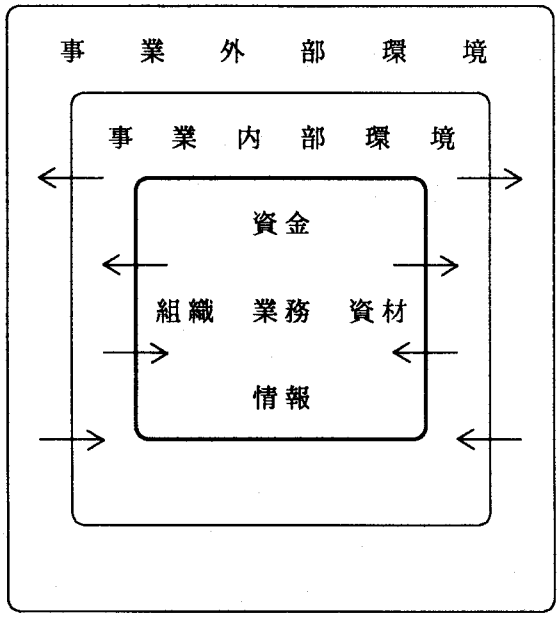
○ バッチ処理環境 － 主として汎用コンピュータ用プログラムの開発時に、計算センターのコンピュータにコンパイル、リンク、テストの一連の作業を行う旨の情報を送り、テスト結果を受け取る。昔は、カードを媒体として原始プログラムと、その後の必要な作業についての情報（ＪＣＬ等）を計算センターの受付に提出し、実行結果をやはりそこから受け取っていた。
ＴＳＳ（１９７０年代中頃以降）の普及後は、端末から、その一連のリクエストを入力してコンピュータに送り、ある程度の時間の経過後に実行結果の情報を見ることが可能となった。

○ ＴＳＳ環境 － 汎用コンピュータや、ミニコンピュータの端末上で、プログラム開発者が直接、エディタ、コンパイラ、リンカ、ローダを起動し、その場で計算結果を確認する。

○ スタンドアローン環境 － ＰＣやＷＳのように個人コンピュータ利用環境においては、これらのプロセスをすべてプログラム開発者が手作業として行う。そういう意味では、ＴＳＳ環境とまったく同様であるが、プログラムの管理等一切が個人によって行われ、自由度の高いフレンドリーな環境を享受することができる。

6．ソフトウェアの開発環境

≪システム化対象≫



ソフトウェア開発を広義に捉えると人、物、資金、情報全てが開発環境となる。

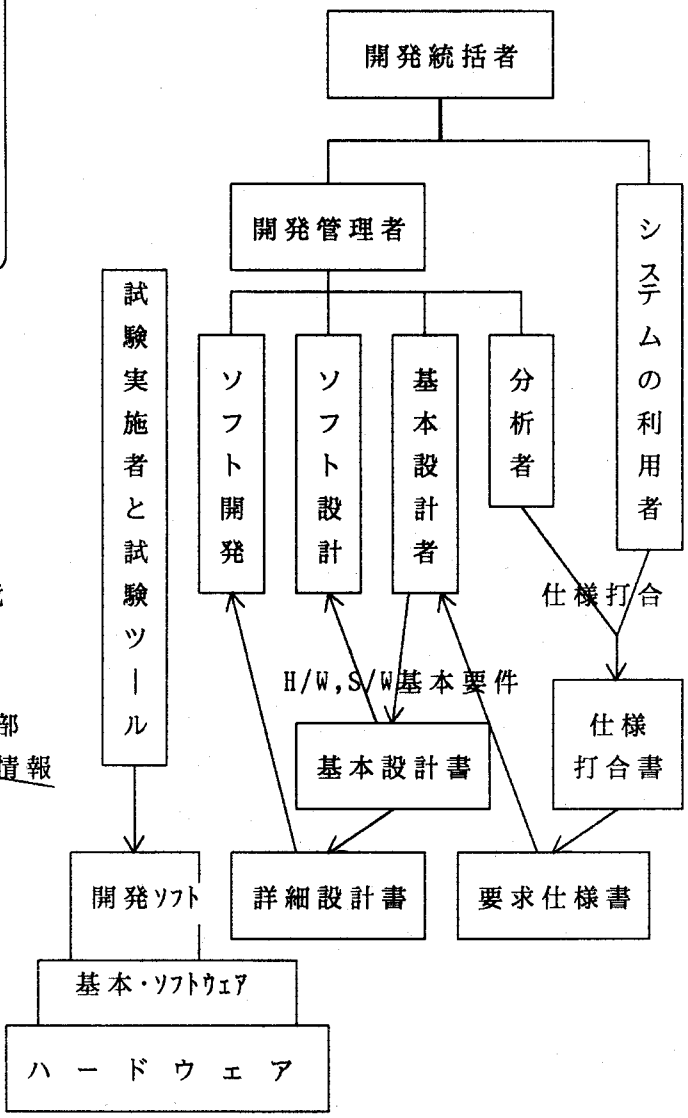
人：利用者、情報技術専門家、開発運用要員

物：コンピュータ、ソフトウェア

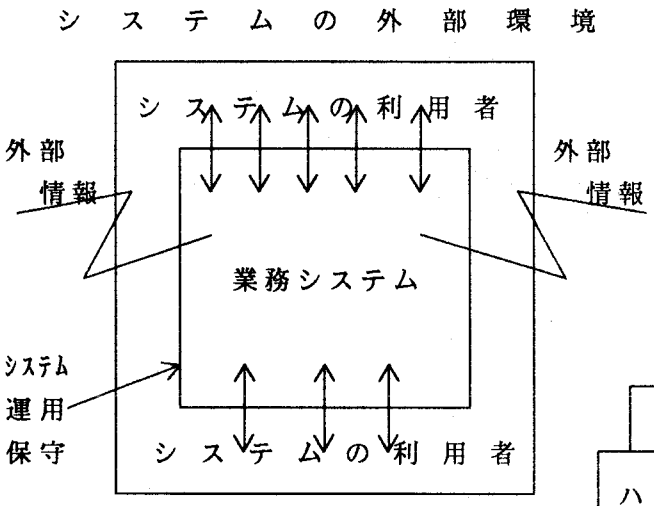
情報：システム目標、技法、技術動向

資金：開発費調達に必要な資金

≪システムの開発≫



≪システムの運用≫



指導上の留意点

一般にソフトウェアは慣れてしまえばさほどむづかしくないことであっても、最初に説明を受けたときはかなり別世界の話を聞かされる心理状態に陥り易い。

ソフトウェアは概して物理的に見えにくいものである。これをできるだけよく目にする光景、実際に提供されているシステムと対比させ、できるだけ分かりやすい例を用いて説明をする必要がある。

パソコンを動かせば、ある程度プログラムが提供する機能については理解ができるであろう。しかし、ソフトウェアの開発プロセスについて理解させるためには、まず、コンピュータの利用状況についてずっと広い範囲で事例的に伝える必要がある。

コンピュータの利用については、産業界（特に、２次産業、３次産業での利用）、研究所、行政における利用例をシステム構成を図入りで提示する必要があろう。指導に際しては、これに相当する資料をコンピュータ関連雑誌や、企業におけるシステム事例などを収集しておく必要もあろう。

また、コンピュータに関する利用場面、技術革新の歴史等についてビデオ等で放映するとイメージが伝わり易い。

第３章　ソフトウェア作成技法の基本事項

指導目標

初めてのプログラミングとの出会いにおいては、若い頭脳ではあっても戸惑うことも多々ありうる。プログラムの組み方に対する理解にも個人差はかなりでてくるであろうが、ソフトウェアの基本的な作成技法を身につけることは必須である。これについても理論づくめで指導するよりは、基本的な処理パターンごとにプログラミング例題をこなしながら学習する方法が最も効果のあるやり方となろう。

基本的な作成技法として、下記の技法についてプログラミング実習することにより、“ちょっとしたプログラム”を書くことができるようになり、プログラムの何たるかを感じとることができよう。高水準プログラミング言語で５０～１００行程度まで作成とテストができるようになることを目標とする。先ずはこの関門をくぐり抜けることが大切である。それ以後は問題がより実務レベルに近づくことにより、アルゴリズム、データ構造、データに対する操作、ファイルの構造、ファイルに関する操作などが複雑になるだけであり、ここで学ぶ基本的な技法の延長といえる。

この章では、コンピュータ・ソフトウェアを作成する上で最も基本となる事項［文書による問題とプログラム、流れ図の利用方法、アルゴリズム、データ処理、ファイル処理］について、最も初歩的な範囲で、時間を掛けて、また実習を通して体に覚え込む。

内容のあらまし

内　　容	説　明	議　論	机上実習	計算機実習
コンピュータによる問題の処理手順	実務的にありうる問題を、コンピュータで解く手順について例を通して知る	特になし	特になし	特になし

内　　容	説　明	議　論	机上実習	計算機実習
流れ図とその利用方法	問題を構造的に把握し、処理手順を詳細に捉えるために流れ図の使い方を学ぶ	同じ問題に対する生徒間の流れ図の表現の違いについて議論させる	小問題を各種選出し、実際に流れ図を書かせる	特になし
問題処理のコンピュータ表現	実務レベルの問題を、コンピュータ処理表現への変換方法と処理方法を学ぶ	流れ図を、コンピュータを余り意識しないレベルと強く意識するレベルとに分けて表現がどう変わるか考える	ブロックチャートレベルの流れ図と、詳細なフローチャート両方書かせる	特になし
計算処理アルゴリズム	各種の計算規則 計算処理構造 基本的算法	特になし	小問題を各種選出し、データ処理の基本パターンに慣れる	汎用コンピュータ、 ＰＣ，ＷＳ
データ処理の方法	主として各種の型の単純変数、および１次元配列を用いてデータ加工法を学ぶ	データを単に数学的な数値としてだけでなく、抽象的な意味を持たせることができることを議論させる	小問題を各種選出し、色々なデータの扱いになれる	汎用コンピュータ、 ＰＣ，ＷＳ
ファイル処理方法	ファイルの基本的な入出力について学ぶ	特になし	ファイルの入出力手続き、エラー時、レコード終了時の処理問題に慣れる	汎用コンピュータ、 ＰＣ，ＷＳ

指導内容

1．コンピュータによる問題の処理手順

コンピュータ処理化したい問題が与えられたときに、どのようなプロセスを経てコンピュータに載せるかを考える。

（1）与えられた問題について吟味する

まず与えられた問題に求められる内容が明確であるか、何らかの形で正確に指示がなされているか、コンピュータにより解が出せるような要求であるかなどを机上で確認する。

① 問題に曖昧な表現がないか確認し、あればそれについてはっきりさせる
② 問題を解くための諸条件が明らかになっているか確認し、欠けていればその不足を補う
③ 計算処理を行う上で、
・何が入力情報となるか
・どのような計算処理を行うか
・計算結果をどのように出力するか
について、詳細に確認しておく。

（2）問題をコンピュータ処理の表現に焼き直す

次にコンピュータ処理化するために、コンピュータ処理における問題の形式に変換する必要がある。

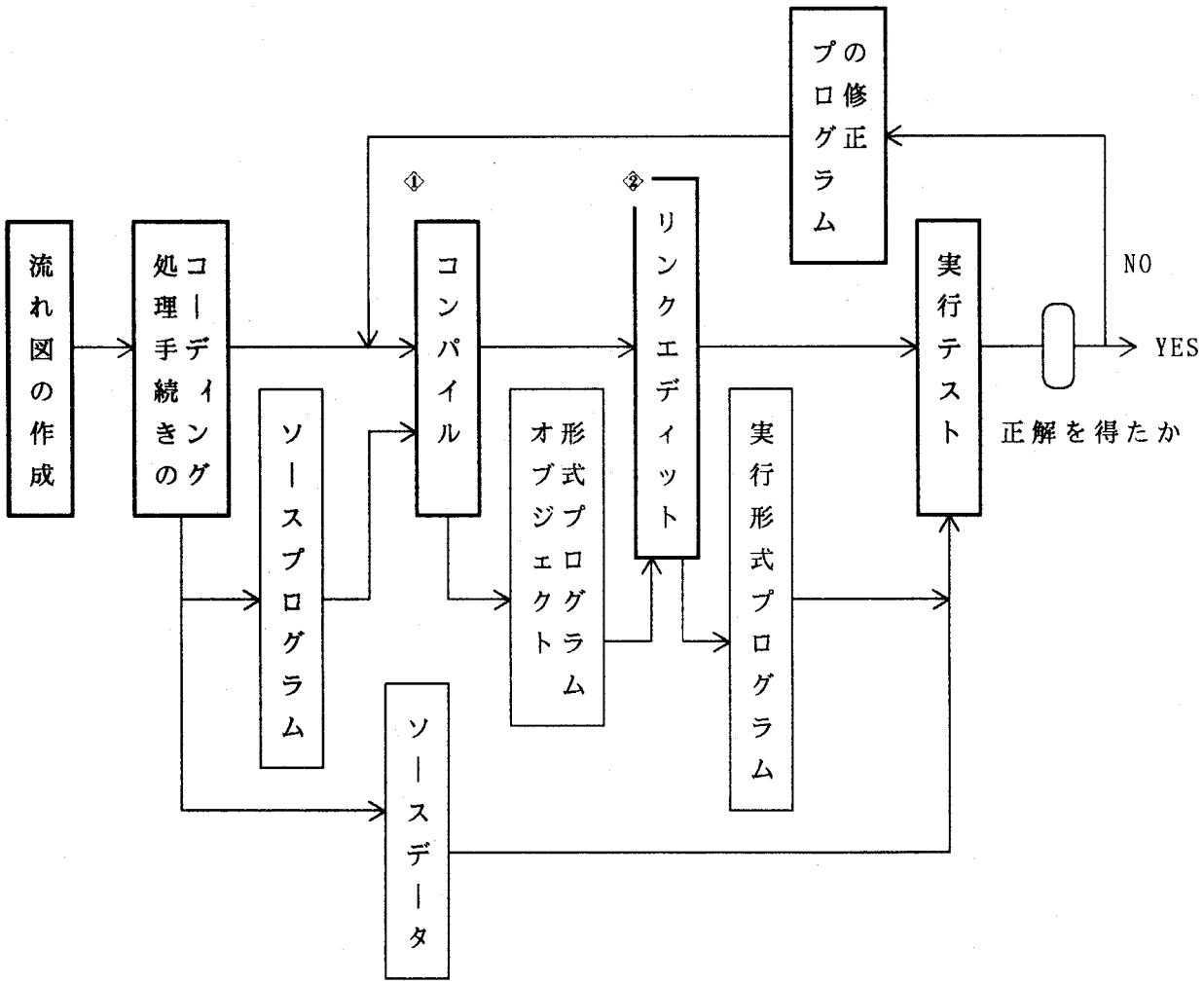
① 全体的な処理手続きはどうなるか
② 計算アルゴリズムを明確にする
③ 問題のコンピュータ表現上どのようなことに留意すべきか

（3）プログラミングとテストを行う

① プログラムの記述言語として何を選択するか
② データをどこに置くか
③ どの程度のメモリ量が必要か
④ ファイルをどう利用するか
⑤ テストをどのように進めるか
⑥ 計算処理の詳細なレベルの過程について流れを明確にする
⑦ プログラム言語でプログラムを書く

⑧　テストの方法について考える
⑨　テストデータを作る
⑩　プログラムを翻訳し、実行形式のプログラムに変換する

≪プログラミング過程≫



①　翻訳という。オブジェクトプログラムのことを目的プログラムとも言う。

②　連携編集という。実行形式プログラムのことをロードモジュールとも言う。

2．流れ図とその利用方法

与えられた問題をプログラム化する前に、流れ図（フローチャートともいう）を書くと計算論理が視覚化され、それが正しいかどうか確認できるとともに、プログラミング前に誤りを発見できる。

ＪＩＳの定義によると、流れ図は、「問題の定義、分析または解法の図的表現であり、データ流れ図、プログラム流れ図およびシステム流れ図」がこれに相当する。以下で簡単にそれぞれの流れ図についてＪＩＳの定義により説明する。

◆ データ流れ図　　問題解決におけるデータの経路を表し、かつ使用する各種のデータ媒体とともに、処理手順を定義する。データ流れ図は以下のものからなる。

- ○ データの存在を示すデータ記号。データ記号は、そのデータを記録する媒体を示すのに用いてもよい。
- ○ データに施される処理を示す処理記号。処理記号は、この処理を行う装置の機能を示すのに用いてもよい。
- ○ 処理やデータ媒体の間のデータの流れを示す線記号。
- ○ データ流れを理解し、かつ作成するのに便宜を与える特殊記号。

◆ プログラム流れ図　　プログラム中における一連の演算を表す。プログラム流れ図は以下のものからなる。

- ○ 実際に行う演算を示す処理記号。論理条件に基づき、それに続く経路を定める記号も含む。
- ○ 制御の流れを示す線記号。
- ○ プログラム流れ図を理解し、かつ作成するのに便宜を与える特殊記号。

◆ システム流れ図　　システムの演算の制御及びデータの流れを表す。システム流れ図は以下のものからなる。

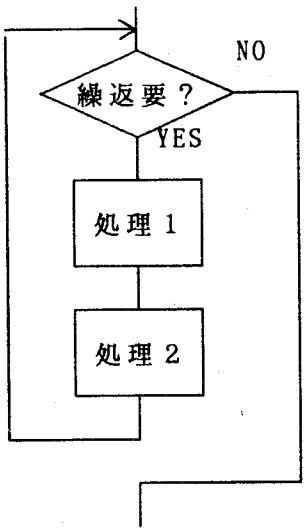
- ○ データの存在を示すデータ記号。データ記号は、そのデータを記録する媒体を示すのに用いてもよい。
- ○ データに施される演算を示したり、それに続く論理経路を定めたりする処理記号。
- ○ 処理やデータ媒体の間のデータの流れを示したり、処理間の制御の流れを示したりする線記号。
- ○ システム線図を理解し、かつ作成するのに便宜を与える特殊記号。

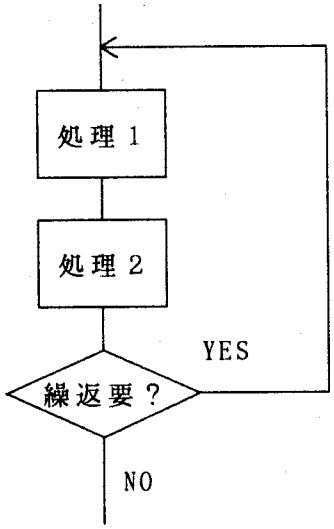
流れ図を作るのに使用する流れ図記号はＪＩＳ［Ｘ０１２１－１９８６］で規定されている。どのような記号を用いるかは、基本データ記号（Ｂasic data symbol）を参照する。これを紙の上に書くときのテンプレートは市販されている。

流れ図を書く場合に留意すべきことは、処理を理解し易いように表現すること、ならびに機能が明確になるよう心がけることである。特に流れの方向については、一定の規則を設けて書かないと処理が読み取りにくくなる。

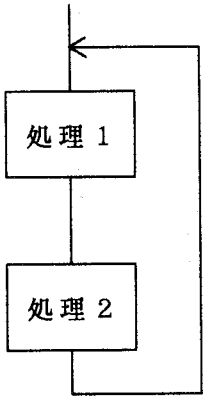
比較的注意したい書き方を以下に示す。

◆　繰り返し（条件つき）





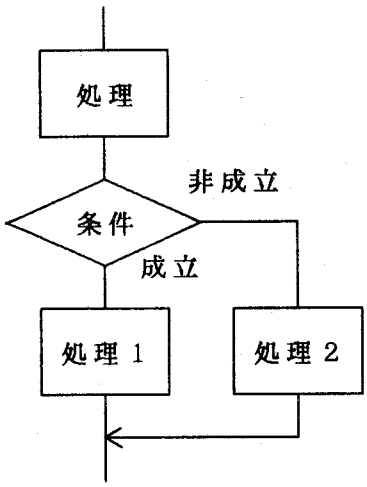
繰り返し（無条件）

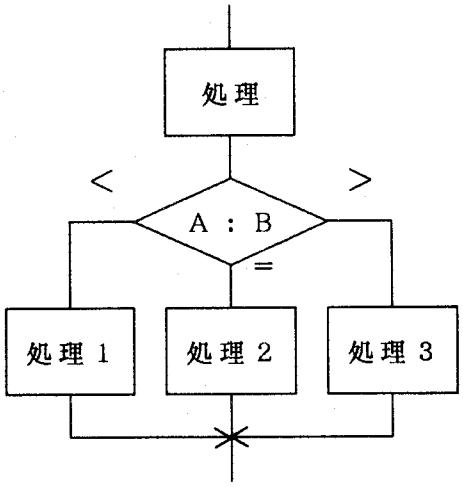


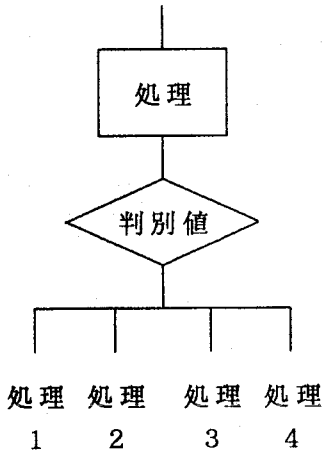
処理１もしくは２において、ある条件の成立によりループから抜けることを、ブレークという

ブレークしない場合永久ループとなる

◆　判断



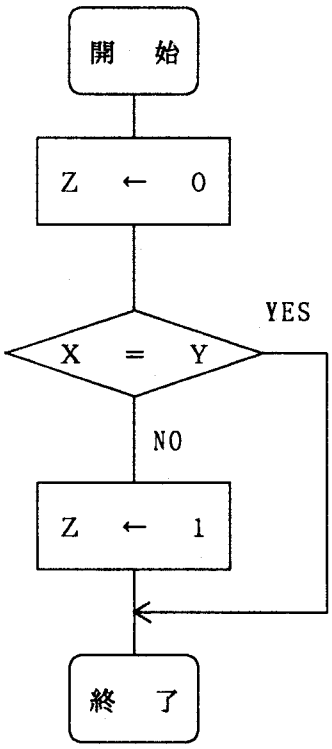




≪流れ図の例題≫

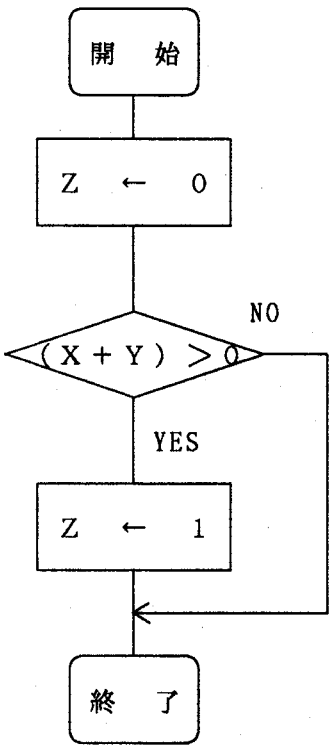
(a) （X, Y）の２つの変数があり、それぞれの値は０もしくは１である。この両変数の排他論理和（変数Ｚに結果をおく）、および論理和を求める流れ図を書く。
これは、アルゴリズミックな処理の流れである。

① 排他論理和



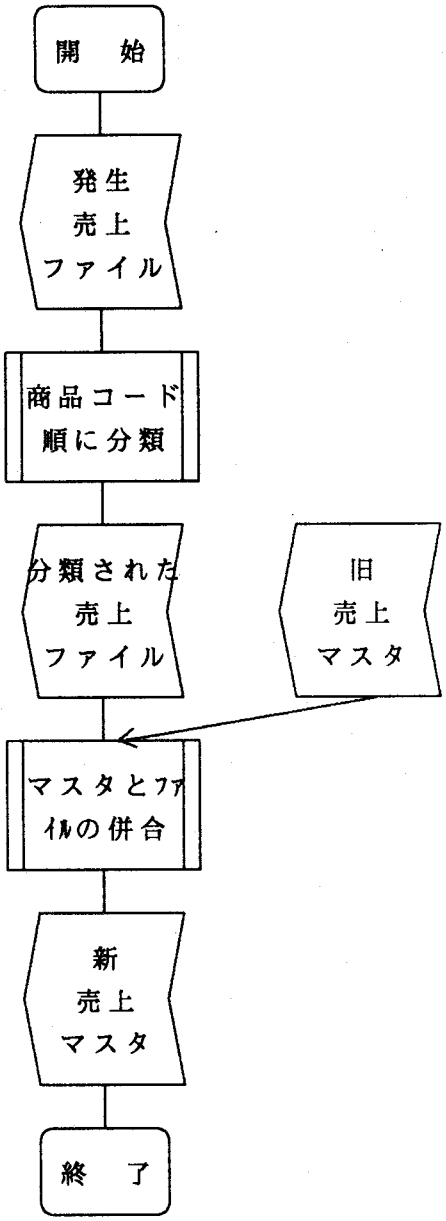
		Y	
		0	1
X	0	0	1
	1	1	0

② 論理和

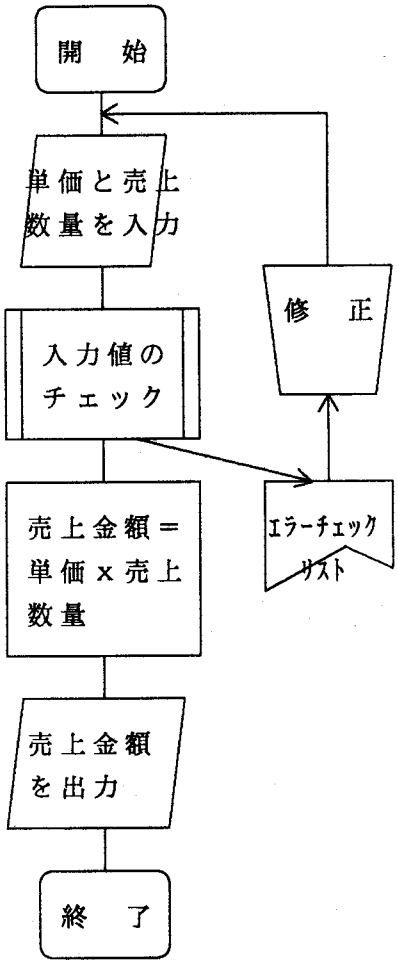


		Y	
		0	1
X	0	0	1
	1	1	1

(b) ランダムに発生した売上データファイルを使って、売上マスタファイルの更新を行う。このシステム流れ図を書く。



(c) 単価と売上数量を入力し、売上金額を求めて、それを出力する。この処理の流れを書く。



3．問題処理のコンピュータ表現

コンピュータ言語によるプログラミングに入る前の段階として、流れ図を書くことを指導したが、問題が簡単なうちはともかく、徐々に複雑になるとともに、いきなり詳細なフローを書くことは、時間効率上良くないし、また誤った処理を起こし易くなる。

プログラミングの資料として、問題の定義書と流れ図だけではなく、コンピュータ化のために問題の種類に応じて、いくつかの文書により処理の詳細を説明する方がプログラマにとっては余程有り難い。これを一般にプログラム仕様書と呼ぶ。

ここではまだそこまで話を及ばないようにするが、コンピュータ処理を行うに際して、机上での説明からコンピュータへと変換されるにつれ、情報処理の世界の言葉や記号により表現することが求められてくる。

一般に詳細（プログラム表現レベルに近い）な流れ図に至るまでに、いくつかの段階を経る必要があることと、流れ図では表しにくいか、もしくは補足となる情報についてまとめる必要がある。

詳細な流れ図を描くまでには、概略の流れをまず明らかにし、処理の大まかな流れを掴むことから始め、次第に処理の詳細に深めていく方法をとる必要がある。また、処理の流れだけが大切なのではなく、さらに処理手順だけでは表しにくいコンピュータ処理もあるため、データ処理の流れ、システム処理の流れなどの観点からも状況を明らかにすることも重要である。

流れを詳細化する過程で、次の事項に注意を払う必要がある。

○　必要な機能を全て抽出し、表現されているか
○　求められる処理性能をどのように確保するか
○　どの程度の主記憶域を必要とするか
○　どのようなデータをファイル化すべきか

また、詳細な処理フローを書く前に、

○　ファイルのとるべきデータ構造について詳細な設計を行う
○　どのような配列をどのくらいのサイズ分使用するか計算しておく
○　どのような表現値を持つ変数を用意すべきか
○　プログラムを使う人とマシンのインタフェースをどうするか

等について、できるだけコンピュータが処理するイメージに近い形で表現できるようにする必要がある。

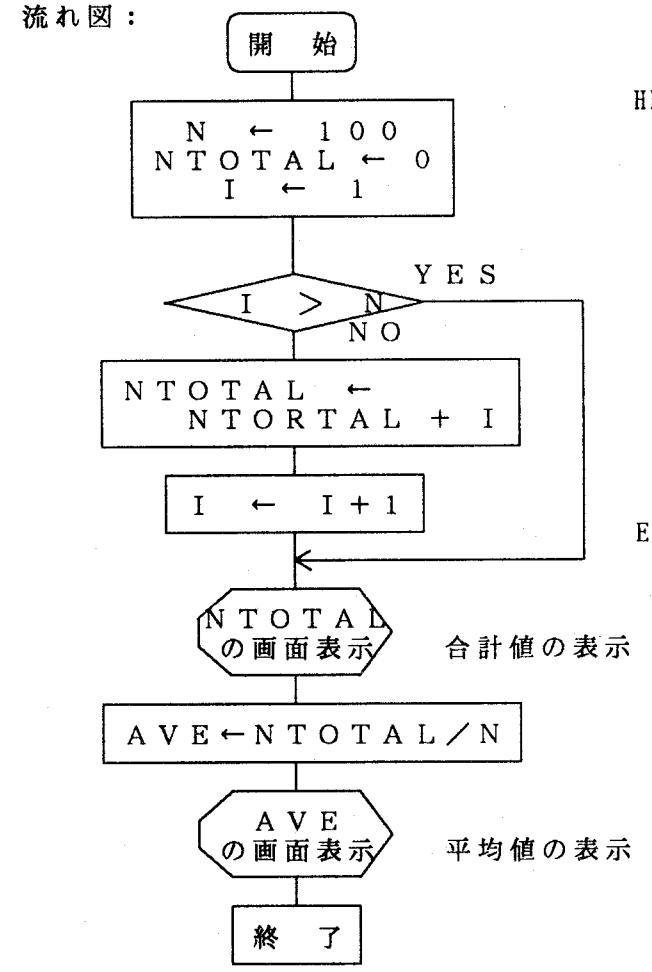
4．計算処理アルゴリズム

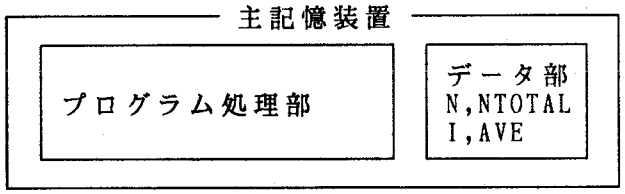
主として、技術計算系の例題を通して流れ図、及び必要に応じてプログラムを用いてアルゴリズムについて指導する。

（１）１からＮまでの自然数の和と平均値を求める

問題：Ｎはプログラム中で予め与えるものとする。
　　計算処理結果として、合計値、平均値を画面に出力するものとする。

流れ図：





［PL/I　ﾌﾟﾛｸﾞﾗﾑ･ﾊﾞｰｼﾞｮﾝ］

```
HEIKIN:PROC OPTIONS(MAIN);

  N      = 100;
  NTOTAL = 0;

  DO I = 1 TO N;
        NTOTAL = NTOTAL + I;
  END;

  PUT EDIT('TOTAL VALUE IS ',NTOTAL)
        (SKIP,A,F(10));
  AVE = NTOTAL / N;

  PUT EDIT('AVERAGE IS ',AVE)
        (SKIP,A,F(10,2));
END HEIKIN;
```

［C　ﾌﾟﾛｸﾞﾗﾑ･ﾊﾞｰｼﾞｮﾝ］

```
main()
{
  int    i,ntotal = 0, n = 100;
  float  ave;

  for ( i=1; i <= n; i++)
  {
     ntotal += i;
  }

  printf('Total Value is %d¥n',
        ntotal);
  ave = (float)ntotal / (float)n;

  printf('Average is %f¥n',ave);
}
```

（２）２次方程式 $a x^2 + b x + c = 0$ の根を求める。

問題： a，b，cの値はプログラム中で予め与えるものとする。
方式としては、判別式 $D = b^2 - 4ac$ を計算し、

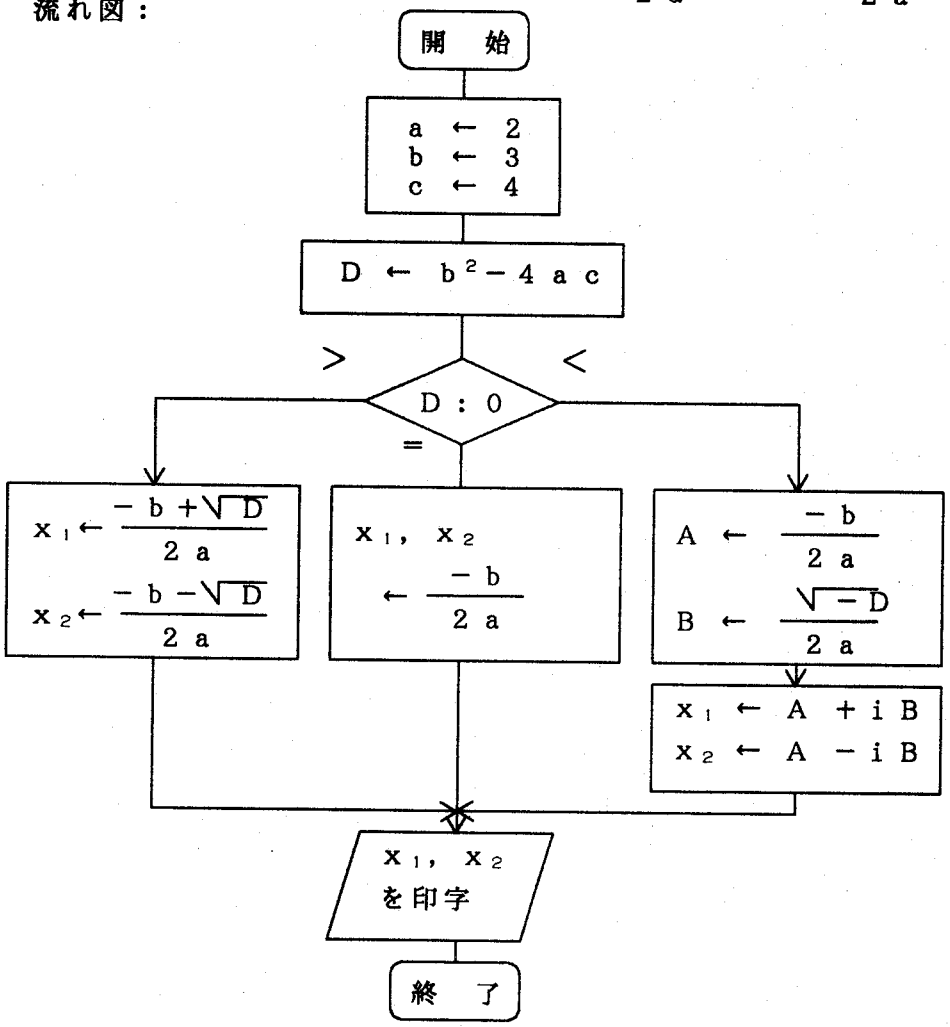
$D > 0$ なら　　$x_1 = \dfrac{-b + \sqrt{D}}{2a}$ ，$x_2 = \dfrac{-b - \sqrt{D}}{2a}$

$D = 0$ なら　　$x_1 = x_2 = \dfrac{-b}{2a}$

$D < 0$ なら　　$x_1 = A + iB$，$x_2 = A - iB$

$\left(A = \dfrac{-b}{2a}\right.$ ，$\left. B = \dfrac{\sqrt{-D}}{2a}\right)$

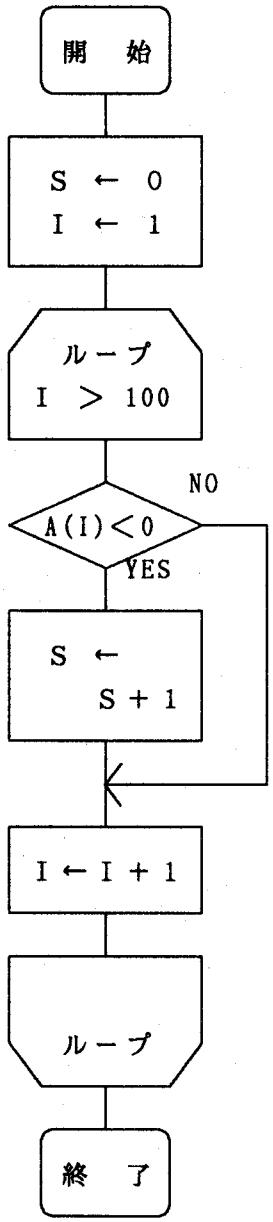
流れ図：



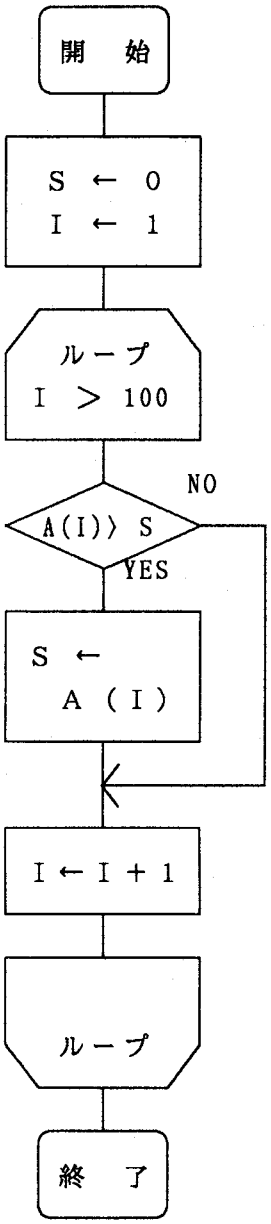
（注）　この問題をプログラミングするときは、ｘを単純変数としてでなく、複素数型の変数として表現する必要がある。例えば、ＦＯＲＴＲＡＮ言語では“COMPLEX”というデータの型が用意されている。

（３）簡単な配列処理

① 配列Ａ（１００）にある負の数の和



② 配列Ａ（１００）にある数の最大値



配列Ａの全ての要素は正の数とする。

上記アルゴリズムにそって、ＦＯＲＴＲＡＮ，ＰＬ／Ｉ，Ｃのいずれかでプログラムを書いてみる。

5．データ処理の方法

ここでは、コンピュータ内部でのデータの表現、コンピュータ言語でのデータの型について解説するとともに、初歩的なプログラムレベルでのいわゆるデータ処理の典型的な例を引用してその概要を知る。

（１）データの表現

機械語やアセンブラ言語のプログラマにはビット、バイト、ワード（特に１６ビットタイプの）データをよく扱う。

これに対して、高水準言語プログラマでは、どのような業務処理のためのプログラミングであるかにより扱うデータの表現形式や構造も変わってくる。

①　ビット表現

コンピュータにおける情報の最小の単位であり、２進数数値として、０または１の値だけを持つ。該当するビット位置の情報がオンであるかオフであるかなど、状態を表すのに多用される。

②　バイト表現

通常連続する８ビットをまとめてバイトと呼ぶ。この単位のデータは、８ビットで現しきれる範囲の整数型の数値を持ったり、またあるときは、アルファベット、特殊な文字、カタカナ等の値を持つ情報として扱われたりする。勿論、データの設計時に８ビット以内の連続するビットに関して、それらのオン・オフ状況に対応してコンピュータ処理上の色々な意味を持たせることも可能である。

③　ワード表現

連続する１６ビットまたは３２ビットをまとめてワードと呼ぶ。この単位のデータは、１６もしくは３２ビットで表しきれる範囲の整数型の数値を持つことが可能である。この場合もバイトと同様、データの設計時に１６（もしくは３２）ビット以内の連続するビットに関して、それらのオン・オフ状況に対応して色々な意味を持たせることも可能である。

④　フィールド表現

複数桁のバイトからなりある意味を持たせた情報となっており、これをフィールドまたは項目などという。

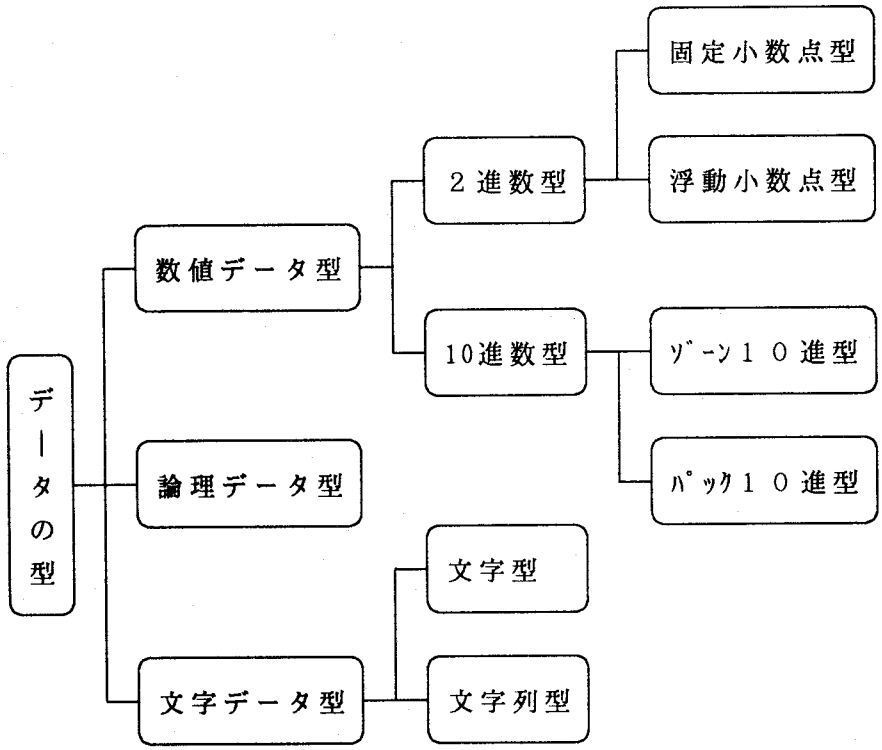
項目には、フィールド全体を代表するタイプと、その中に定義されるサブフィールド的な項目もある。コンピュータ言語ではこの種のデータについてしばしば“構造体”と呼んでいる。

⑤　レコード

フィールドがいくつか集まって構成される情報であり、事務系のデータ処理を行うときの意味的なものと、入出力の単位ともなる。

（２）高水準言語におけるデータの型と形式

①　データの型（タイプ）



○　数値データ型については、“ハードウェアの基礎”編を参照されたい。

○　論理型とは、“真”か“偽”かを現すデータで、高水準言語的なデータ表現である。このデータに対する演算として、論理積、論理和、排他的論理和、論理否定などがある。

○　文字データ型には、１つの文字のみを指す場合と、連続する文字列を指す場合とがある。

②　データの形式

このような型のデータは下記形式で記憶装置上に配置される。

○　定数として記憶装置上に置かれる。（データ領域として）

○　変数として記憶装置上に置かれる。（変数領域として）

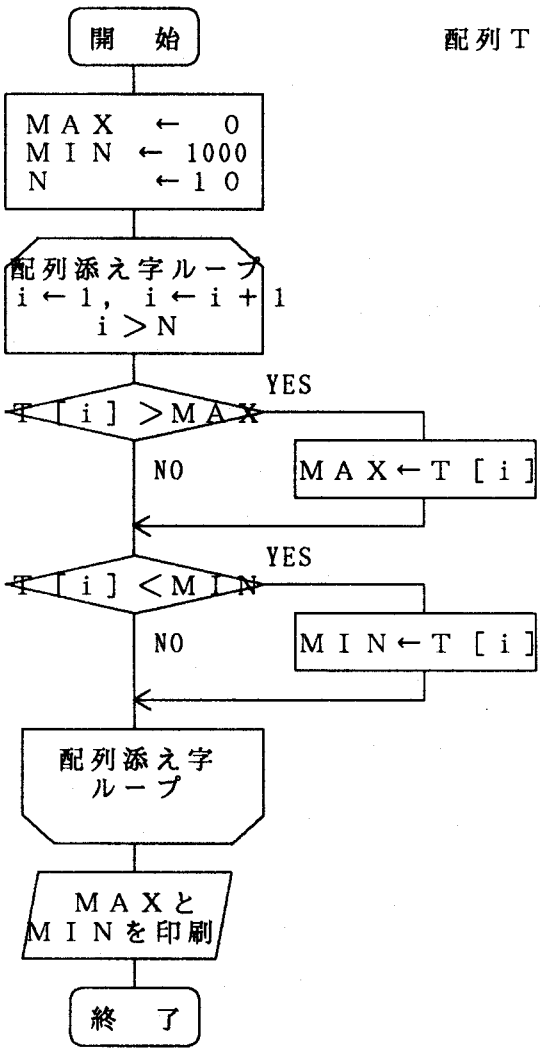
○　連続する同じ型の変数領域として配置される配列

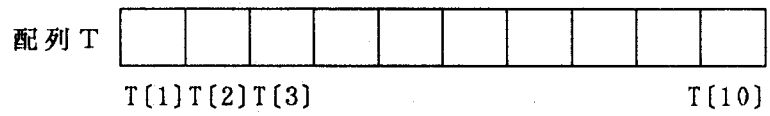
○　いくつかの型のデータがセットになった情報で構造体と呼ばれるものがある。構造体自身もまた配列を構成することもできる。

（３）配列を扱う処理

① 配列に置かれているＮ個（例では１０とする）の値（３桁以内の正数）の内、最大値と最小値を求める。

配列Ｔには以下のようにデータが既に入っているものと仮定する。

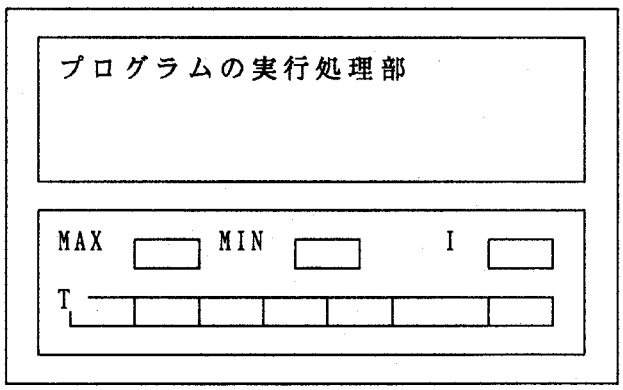




［ＰＬ／Ｉバージョンプログラム］

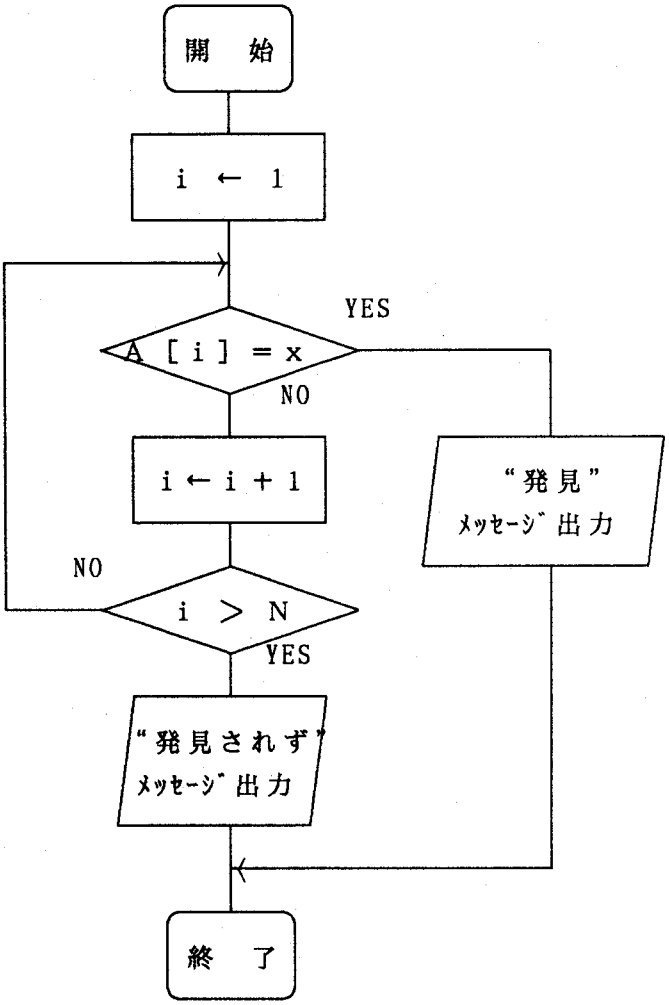
```
MAXMIN:PROC OPTIONS(MAIN);

  DCL T(10) BIN FIXED(15)
  INIT(30,20,15,40,50,35,5,25,10,50);

  MAX = 0;  MIN = 1000;  N = 10;

  DO I = 1 TO N
    IF MAX < T[I] THEN  MAX = T[I];
    IF MIN > T[I] THEN  MIN = T[I];
  END;

  PUT LIST  (MAX,MIN);

END MAXMIN;
```

主記憶装置上のプログラム



ＰＬ／Ｉコンパイラは、原始プログラムをコンパイルすると、
・手続き部
・データ部
に分離する。データ部は、変数や配列などについて、必要領域を確保し位置を決める。

② Ｎ個の要素からなる配列Ａにランダムな順にデータが入っているものとする。
特定の値Ｘがその配列の中に存在するかどうかを探してみる。



配列を初めから順に見ていき、見つけたらこの操作を終了する。データが全くバラバラに入っている場合、この方法を用いる。

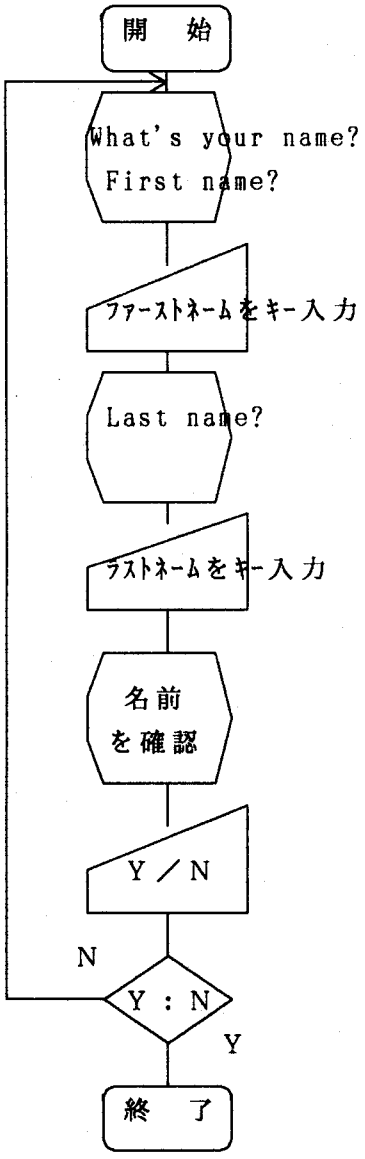
必ず見つかるものとすると、Ｎ個の要素を持つ配列を探す場合、平均（Ｎ＋１）／２回、最大Ｎ回の繰り返しで見つけられる。

左記フローにつき、［ＰＬ／Ｉ、ＦＯＲＴＲＡＮ、Ｃ］のいずれかにより、プログラムを作りなさい。

（４）画面との対話を行う処理

生徒がコンピュータ画面上で以下のやりとりを行うプログラムを作る。

・画面（プログラム）から生徒に対して、氏名を聞いてくる
・最初にファーストネームを入れる
・次にラストネームを入れる
・入力した名前が正しかったかどうか尋ね、正しければ“Ｙ”を、正しくなければ“Ｎ”を入力する。
・正しくなければ、最初から同じことを繰り返す。



ＰＬ／Ｉバージョン・プログラム

```
INTRACT:PROC OPTIONS(MAIN);

  DCL  FIRST   CHAR(20) ,
       LAST    CHAR(20) ,
       ANS     CHAR(1) INIT('N');

  DO WHILE (ANS='N');

    PUT SKIP LIST('WHTAT'S YOUR NAME?');
    PUT SKIP LIST('ENTER FIRST NAME:');

    GET LIST(FIRST);

    PUT SKIP LIST('ENTER LAST NAME:');

    GET LIST(LAST);

    PUT EDIT (
          'THANK YOU,SO YOUR NAME IS',
          FIRST,LAST,
          ', ISN'T IT? -- Y/N');
          (SKIP,4A);

    GET LIST (ANS);

  END;
END INTRACT;
```

ＦＯＲＴＲＡＮバージョン

空きの部分を埋める。
下記ステートメントから選ぶ。

```
      CHARACTER*20   FIRST , LAST ,
                     ANS
      DATA  ANS/'N'/
C
   10 WRITE(6,1000)
      READ(5,2000) FIRST
C
      WRITE(6,1100)
      READ(5,2000) LAST
C
      WRITE(6,1200) FIRST,LAST
      READ(5,2100) ANS
C
      ┌──────────────────────┐
C     └──────────────────────┘
   20 STOP
C
 1000 FORMAT(' WHAT'S YOUR NAME?'/
     '  ENTER FIRST NAME?')
 1100 FORMAT('1ENTER LAST  NAME?')
 1200 FORMAT('THANK YOU,',
     1       'SO YOUR NAME IS ',
     2       A20,2X,A20,
     3       ', ISN'T IT? -- Y/N')
 2000 FORMAT(A20)
 2100 FORMAT(A1)
C
      END
```

㋐ IF (ANS .EQ. 'Y') GOTO 10
㋑ IF (ANS .NE. 'Y') GOTO 10　正解
㋒ IF (ANS .EQ. 'Y') GOTO 20

Ｃバージョン

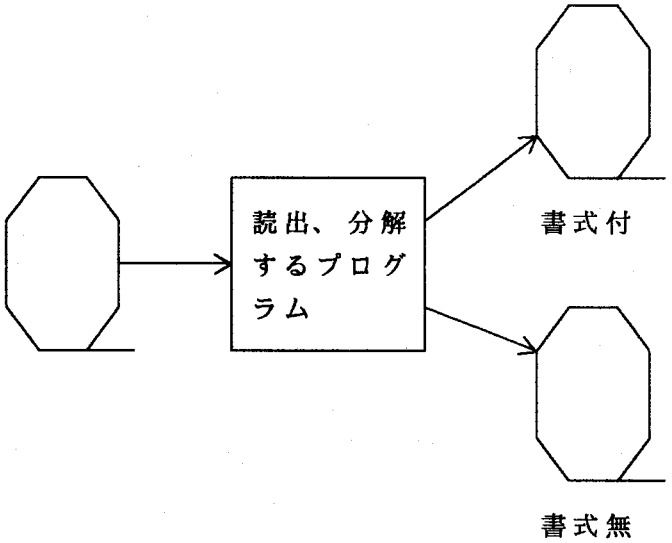
空きの部分を埋める。
下記ステートメントから選ぶ。

```
main()
{
  char    first[21],last[21],
          ans[2];

  ┌──────────────────────┐
  └──────────────────────┘
   {
      printf("What's your name?¥n",
             "Enter First name: ");
      scanf("%s",first);

      printf("¥nEnter last name: ");
      scanf("%s",last);
      printf("¥nThank you,¥
so your name is %s¥
,isn't it? -- Y/N");
      scanf("%s",ans);
   }
}
```

㋐ for (ans[0]='N'; ans[0]!='N';)
㋑ for (ans[0]='Y'; ans[0]!='N';)
㋒ for (ans[0]='N'; ans[0]=='N';)
㋓ for (ans[0]='Y'; ans[0]=='N';)

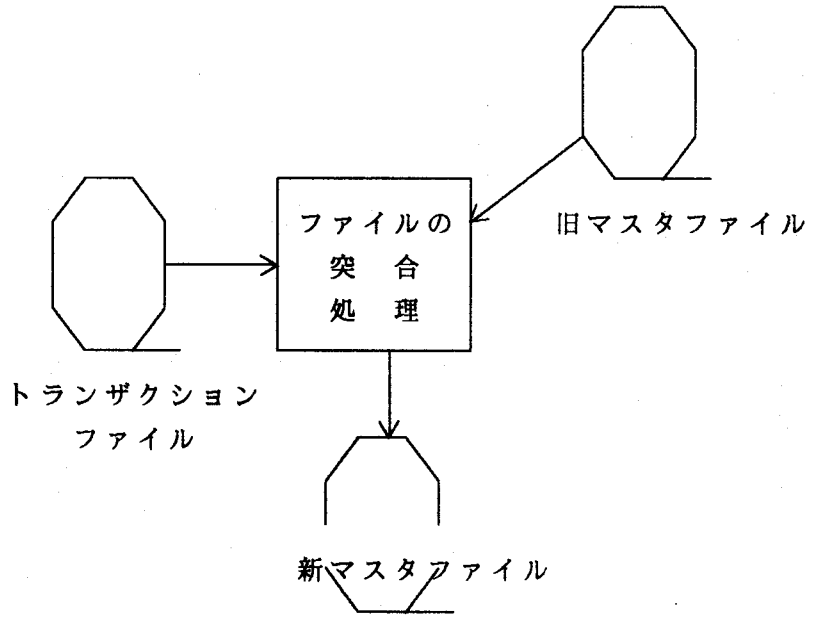
6．ファイル処理方法

（１）１つのファイルを読み込み、レコードの内容により２種類のファイルに分割し、別々の媒体に書き出す。




```
      CHARACTER    A(76)
C
   10 READ(1,100,END=999)  I,A
C
      IF ( I .EQ. 0 )  THEN
        WRITE(3)          I,A
      ELSE
        WRITE(2,100)      I,A
      ENDIF
C
  100 FORMAT(I4,A76)
C
  999 STOP
      END
```

（２）トランザクションファイルレコードとマスタファイルとの突合により、併合して新しいマスタファイルを作成する。

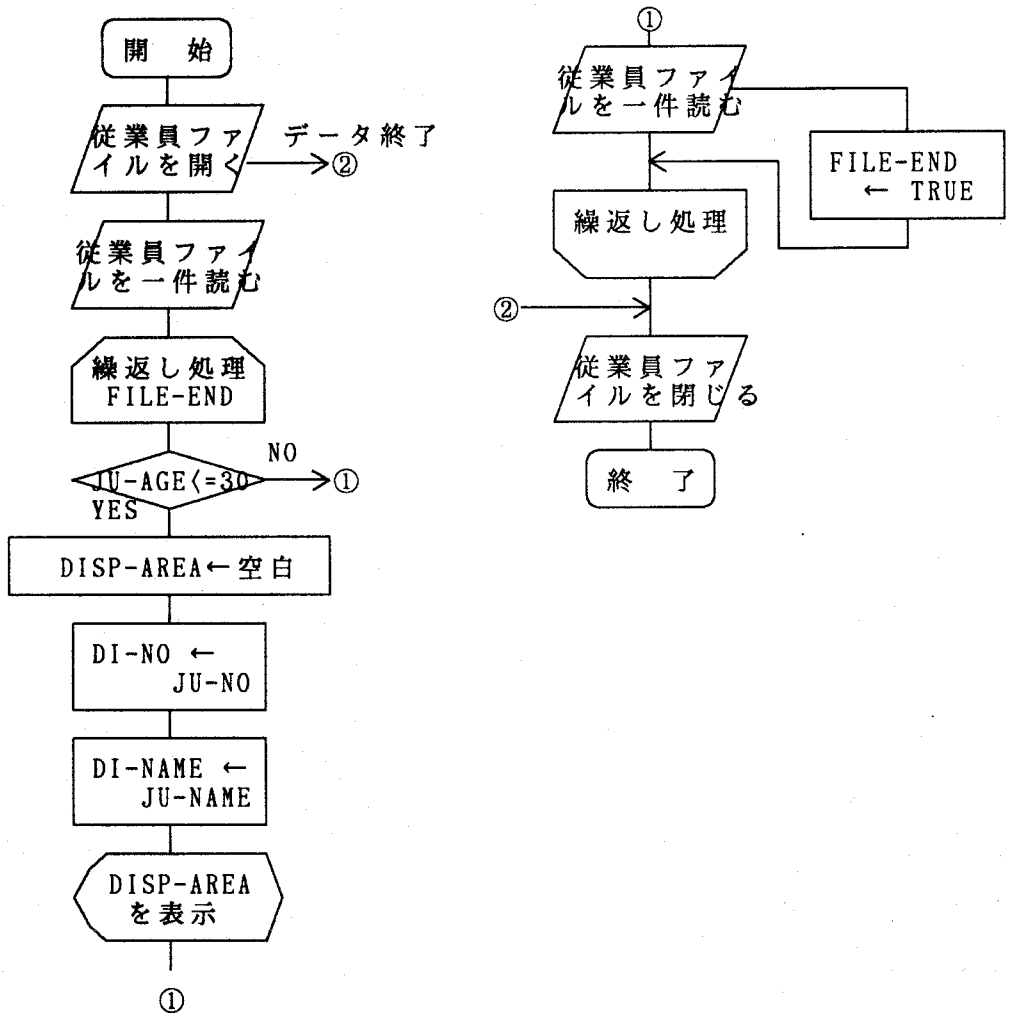


（３）従業員ファイルから、年令３０歳以下の従業員を選び、従業員番号と名前を表示する

従業員ファイルのレコード様式は、下記の通り。

従業員番号 ４桁	名前 ２０桁	年令 ２桁	その他 ５４桁

順編成の従業員ファイルを一件ずつ順に読み、年令が３０歳以下のレコードを拾う。入力データがなくなったら処理を終える。



[COBOLプログラム]

```
DATA                          DIVISION.
FILE                          SECTION.
FD   JUGYOIN-FILE.
01   JUGYOIN-REC.
       10  JU-NO                    PIC  9(4).
       10  JU-NAME                  PIC  X(20).
       10  JU-AGE                   PIC  9(2).
       10                           PIC  X(54).
WORKING-STORAGE          SECTION.
01   END-SW              PIC  X  VALUE  LOW-VALUE.
       88   FILE-END                VALUE  HIGH-VALUE.
01   DISP-AREA.
       10  DI-NO                    PIC  9(4).
       10                           PIC  X(10).
       10  DI-NAME                  PIC  X(20).
PROCEDURE                     DIVISION.
HAJIME.
       OPEN  INPUT  JUGYOIN-FILE.
       READ  JUGYOIN-FILE  AT  END  GO  TO  OWA.
       PERFORM  UNTIL  FILE-END
          IF  JU-AGE  <=  30
             MOVE  SPACE     TO  DISP-AREA
             MOVE  JU-NO     TO  DI-NO
             MOVE  JU-NAME   TO  DI-NAME
             DISPLAY  DISP-AREA
          END-IF
          READ  JUGYOIN-FILE
             AT  END  SET  FILE-END
                      TO  TRUE  END-READ
       END-PERFORM.
OWA.
       CLOSE  JUGYOIN-FILE.
       STOP  RUN.
```

指導上の留意点

与えられた問題がどうコンピュータ表現されるべきかについては、プログラミングについてのある程度の知識が必要になる。しかし、その一方で、ソフトウェアの入門者がプログラミング上の約束ごとを一度に押しつけられると、理解に時間がかかる上に、コンピュータ処理による表現の本質をなかなか掴みにくい状況にもなる。

そこで、種々ある高水準プログラミング言語に共通する事項を最初に教える必要がある。例えば、

○　データの表現（変数の型や、表現できる値の範囲など）
○　変数の参照、変数への値の代入、変数に対する演算
○　変数への値の読み込み
○　変数への値の設定の意味（例：－１→下、０→中、１→上、
　　あるいは０→男性、１→女性　など）
○　配列の記憶装置上での並び方
○　画面からの入力や画面への出力

は必須の事項である。

その後、種々の問題について流れ図をどんどん書かせる。この場合、問題としては、あまり言語の種類に依存しないものを選んだ方が混乱が起きない。

流れ図の書き方に十分慣れた段階で、簡単なプログラミングにはいる。プログラミングについては、とにかく問題例に数多く触れさせることがソフトウェアを理解する上での最も近道である。

第４章　プログラミングとテストに関連する事項

指導目標

自力でコーディングしたプログラムについて正解を得られなかった場合、まず自ら誤りを発見する態度と、そのコツをつかむことが重要である。また、プログラミングの初心者に限った話ではないが、テスト能力の限界からとかく誤りの原因はマシンにあると断定しがちである。

一般に、コーディングしたものにはバグが少なくあるべきことは当然であるが、プログラミング初心者にとってはむしろ誤りがたくさんあり、その度にデバッグを繰り返す方が色々なことを体験し、また物事を考える時間を多く持つことができ、より好ましいことでもある。このプロセスこそプログラミングのより高い能力を養うものと想定した学習を進めるべきである。

次に、サブルーチンの必要性を早くから認識させることも非常に重要である。初心者はだらだらプログラムを書きがちである。

内容のあらまし

内　　容	説　明	議　論	机上実習	計算機実習
プログラムの基本的なデバッグ方法	プログラムのデバッグについて、問題理解の再確認、コードのレビュー方法、コーディングミスの推定、デバッグライト、デバッガの利用法を理解させる。	特になし	間違いのある原始プログラムを提示し、誤りを指摘させる。	汎用コンピュータ ＰＣ，ＷＳ デバッグ文の使用、デバッガの使用を含める
サブルーチンについて	プログラミングにサブルーチンを導入する。 サブルーチンの役割を学ばせる	サブルーチン化のメリットについて、議論させる。	分割が好ましいプログラムを提示し、サブルーチンを作成させる。	汎用コンピュータ ＰＣ，ＷＳ
モジュール分割と結合	モジュール分割の必要性、モジュールの構成法、モジュール結合について学ばせる。	実際に生徒のプログラムについて、適切な分割になっているか議論させる。		汎用コンピュータ ＰＣ，ＷＳ
モジュール結合プログラムのテスト	結合されたプログラムのテスト方法について学ばせる。	特になし	比較的大きなプログラムをチームを組んでプログラミングおよびテストさせる	ＰＣ，ＷＳ

指導内容

1．プログラムの基本的なデバッグ方法

プログラムのデバッグに関しては色々な手段が考えられる。一般にプログラムの大きさが小さいほどバグは発生しにくく、またデバッグも楽である。大きなプログラムほど不具合の数が多くなるとともに、不具合を修正するのもずっと困難になる。

コンピュータの歴史が進むとともに、プログラムは巨大化してきている。このため、プログラムの開発者は、バグを迅速かつ確実に発見する方法を追究してきた。例えば、

・不具合をできるだけ減らすために、高品質で網羅性の高いテスト方法を考え出してきた
・人力によるバグの発見や解決を行う前に、できるだけコンピュータのソフトウェアを使って自動的にバグを発見しようという試みと、その成果を得てきた

等が現在、ソフトウェア工学という一種のソフトウェア学問にまで発展している。

プログラムの初心者にとって、いきなり大きなプログラムを作ることはないはずであり、通常の規模のプログラム・デバッグを行う上で参考となる注意事項について以下で述べる。

デバッグの種類	デ　バ　ッ　グ　の　内　容	留　意　点
①仕様の確認	まず、問題を正しく捉えたか、問題をコンピュータ処理の表現に正しく変換したか等を確認する。問題が正しくない、適切な表現になっていないこともありうる。この仕様確認はプログラミングにはいる前には必ず実行するが、それ以降も必要に応じ実施する。	問題を出した人と十分に仕様につき話し合うことが重要。
②目視チェック	原始プログラムを目視で、コーディングに誤りがないかどうかをチェックする。リンクや実行に入る前、またうまく実行動作しないときに、ソースコードに立ち戻ることになる。この方法でかなりのミスが発見される。プログラムの規模に関わらず共通して習慣化すべき行為であろう。	プログラムの規模が大きくなると、他人のチェックも重要となる。
③実行結果からの誤り推定	出来上がったプログラムを実行すると、正常に動作しない場合にはその時点での情報が得られることがある。例えば、実行の途中までの結果、異常終了したコード、出力結果の値の異常等がある。 ・配列の上限、下限オーバ ・変数の初期化の忘れ ・変数名の使い間違い ・数値オーバフロー ・アベンド(Abnormal end)リストの情報	プログラムのデバッグはできるだけ実行結果から状況を判断し、過ちがどこにあるかを論理的に突き止めることが肝要。

デバッグの種類	デ　バ　ッ　グ　の　内　容	留意点
④デバッグ文、デバッグライトの埋め込み	実行結果からだけではなかなか不具合の原因がどこにあるか分からない場合がある。このようなときに、原始プログラムの処理部の中で途中結果をプリントし、どこで誤りが発生し始めたかを発見することに役立てる。 通常、変数の状態や配列の中身をプリントする。 即ち、変数や配列のスナップ、トレースを行う。 多用すると、出力情報が多くなりすぎ、かえって混乱を起こすこともありうる。 効率的に情報をプリントするために、外から指示した変数や配列に関する情報だけを取れるよう工夫することが望ましい。	デバッグが終了したら、デバッグライト文は通常削除する。重要なデバッグ文については、残す場合もある。
⑤デバッガの利用	①～④では、どうしてもデバッグが困難なときにデバッガのお世話になる。デバッガとしては、２通りの利用方法がある。 (a) インタプリータの利用 ある種類の高水準言語では、コンパイラの他にインタプリータも用意されている。この場合には、インタプリータで実行動作を確認しておき、主要部分が正常に動くと確認されたら、ロードモジュールレベルで実行を行うことができる。 (b) シンボリックデバッガの利用 このデバッガの制御のもとで対象プログラムをデバッグする。（低水準、高水準どちらも適用可） ・トレース機能 ・スナップショットダンプ機能 ・メモリダンプ機能 ・ファイルダンプ機能	インタプリータは、主として、高水準言語の初心者、あるいは、非専門的な立場にある人がプログラミングを行う場合に有効である。 机上デバッグ等により、手に負えない重要で細かな誤りに適用する
⑥静的解析ツールの利用	コードオーディタ等と呼び、原始プログラムを対象として、その中の誤りや、おかしな挙動を起こしそうな部分を発見するソフトウェアツールである。 ・変数の初期化の忘れ ・絶対に実行されない処理部分 ・デバッグの必要なテストケースの抽出 ・配列の上下限範囲外使用	静的解析（コードオーディタ）ツールは大規模なプログラムの全体テストを行うときに有効

2．サブルーチンについて

プログラムを構成するとともに、コンパイルの基本単位となる情報の集まり（プログラムステートメント群）をルーチンと呼ぶ。

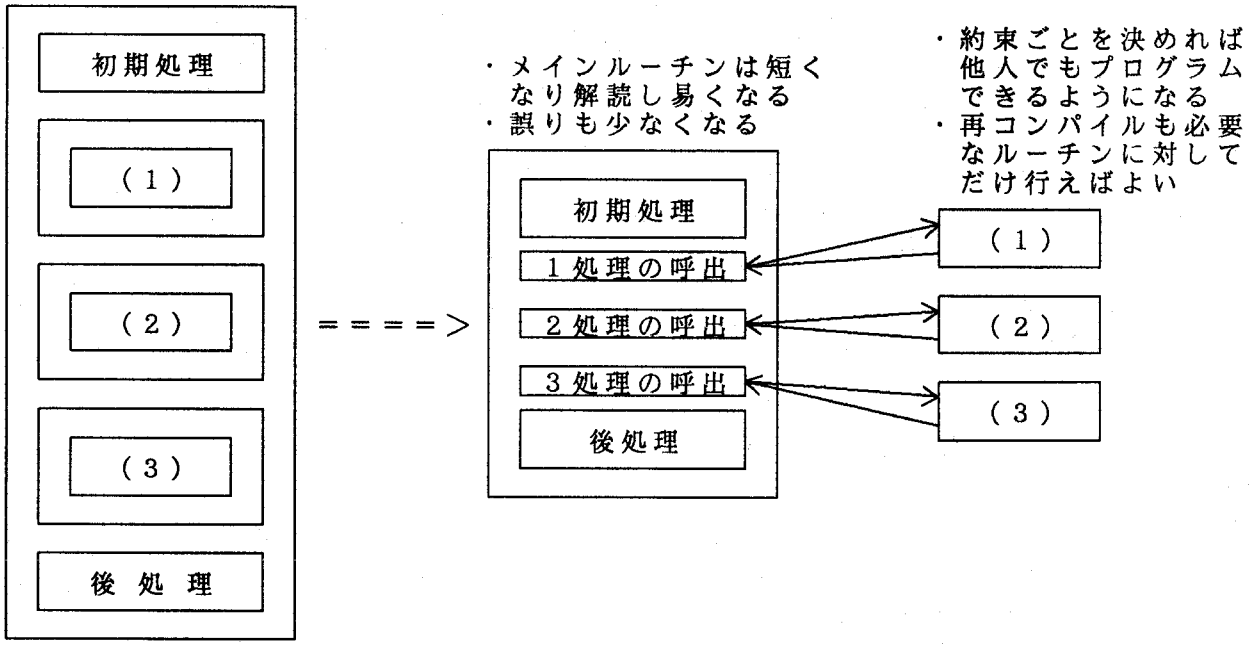
このルーチンにはメインルーチンとサブルーチンとがあり、前者は、リンクエディットして実行形式プログラムを作るときには必ず、かつただ一つ含むことができるもので、実行開始後にオペレーティングシステムの制御プログラムからエントリしてきて、最初に処理される部分であり、また、プログラムの終了する部分を持っている。

また、サブルーチンとは、メインプログラム同様に一連の処理手続きとデータからなる点で変わらないが、上位のルーチン（メインルーチンや他のサブルーチン）からコールされるものである。他のいくつかのルーチンで必要とする共通的な手続きを行うときにしばしば作成される。

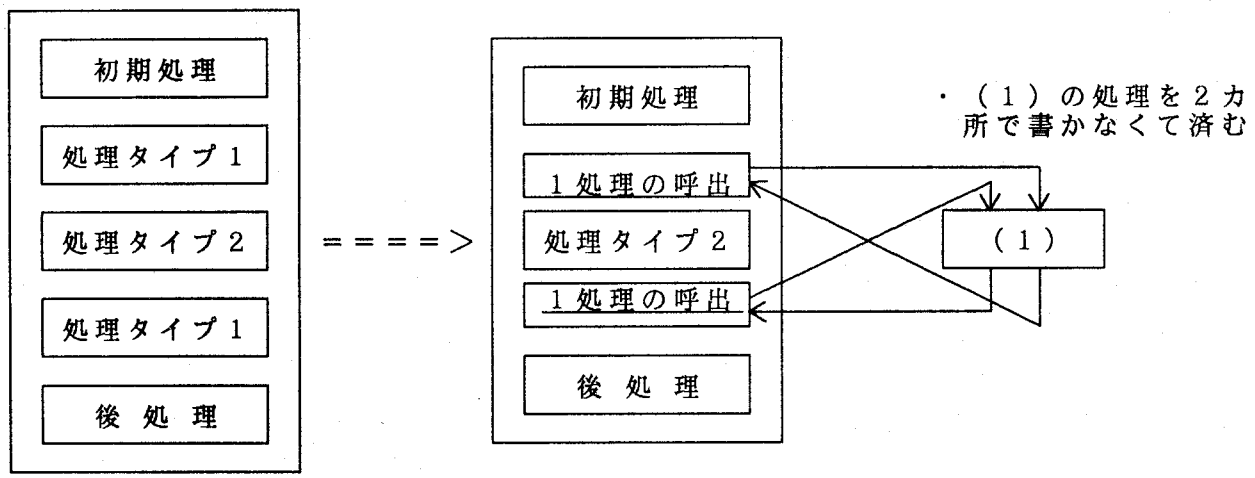
この“ルーチン”は、プログラミング上の言葉であり、オペレーティングシステムのファイル管理（後述）概念から捉えると、モジュールと呼ばれるのが通常である。

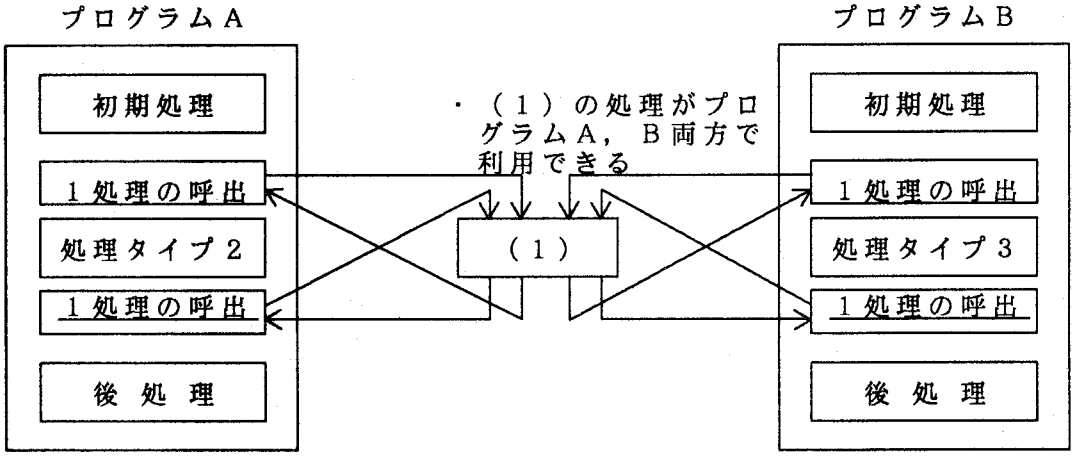
サブルーチンはプログラムを作り始めれば自然にその必要性を感じるようになるものではあるが、どのような処理部分をどのようにサブルーチン化するかはプログラミングの慣れを要するものである。以下で、まず一般論としてサブルーチンを作る場面について考えてみる。

① プログラム表現の簡素化

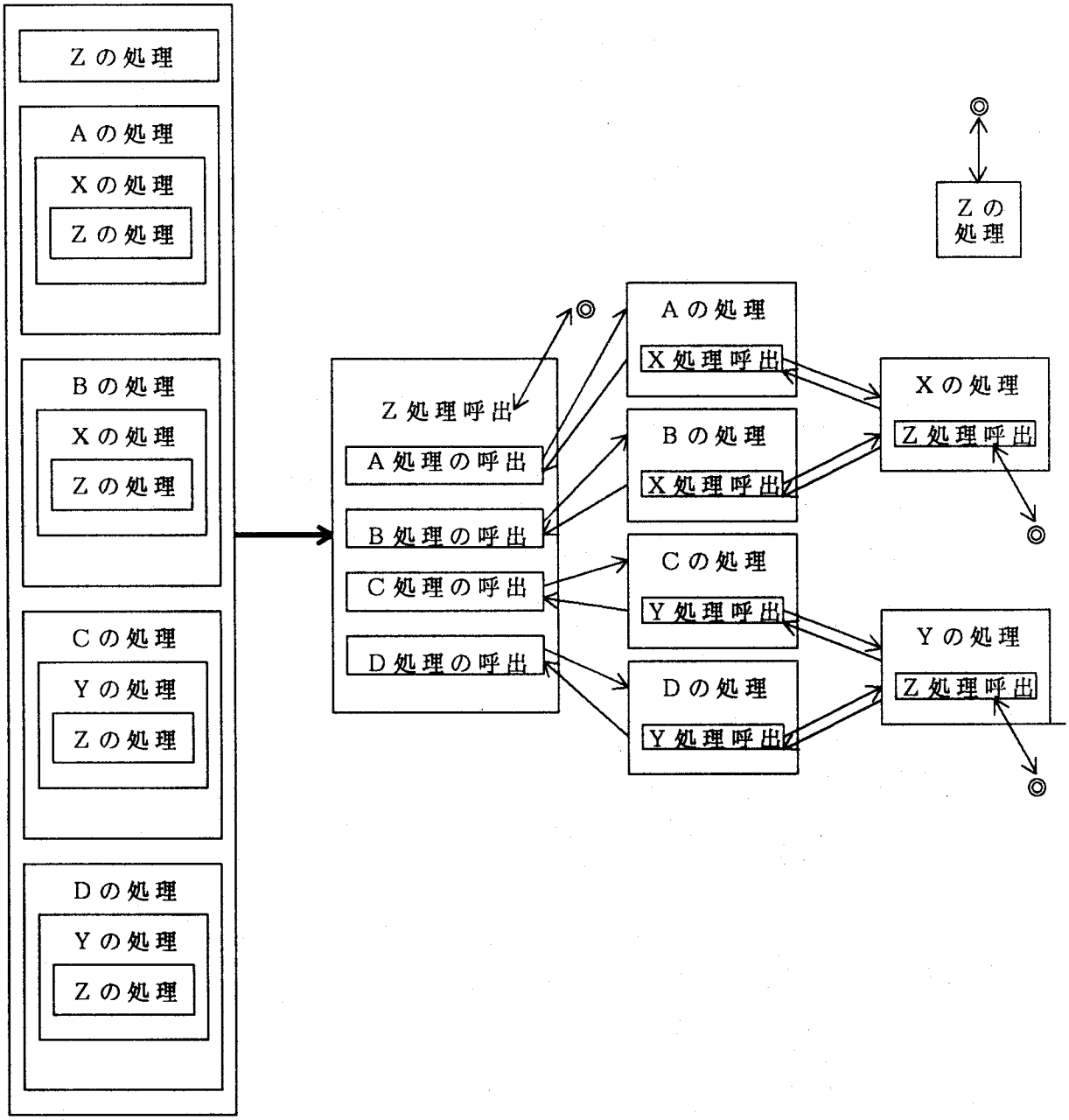


② プログラム処理手続きの共通化





③　より深いサブルーチン



④　サブルーチンのプログラム例

下記のような種類のデータを入力（入力エラーはないものとする）し、それぞれの値が有効であるかどうかを確認するプログラムを作る。〔各入力データの個数は１０以下とする。また、値は４桁以内とする。〕

(a)

k	ﾃﾞｰﾀ1	ﾃﾞｰﾀ2	ﾃﾞｰﾀ3	・・・・	ﾃﾞｰﾀk

（ｋは実効データの個数）

各データの値の限界値についてはプログラムで分かっているものとする。

(b)

l	ﾃﾞｰﾀ1	ﾃﾞｰﾀ2	・・・・	ﾃﾞｰﾀl

（ｌは実効データの個数）

各データ(非０)の正しい符号についてはプログラムでわかっているものとする。

(c)

m	ﾃﾞｰﾀ1	ﾃﾞｰﾀ2	・・・・	ﾃﾞｰﾀm	ﾊﾟﾘﾃｨ値

データ１～データｍは各々値が０または１であり、末尾は偶数パリティー値である。

(d)

n	ﾃﾞｰﾀ1	ﾃﾞｰﾀ2	ﾃﾞｰﾀ3	・・・・	ﾃﾞｰﾀn	合計値

末尾にデータ１～データｎの手計算での合計値がある。

［ＦＯＲＴＲＡＮバージョン］　－　MAINプログラムだけで処理するケース

```
      INTEGER*4   COUNT,VALS(10),TAIL
      INTEGER*4   VLIMIT(10),VSIGN(10)
      CHARACTER*4 ERRLIN(10)
C
      DATA        VLIMIT/40,90,60,20,80,70,50,10,30,100/      ･･･ 限界値検査用
     1            VSIGN /-1, 1, 1,-1, 1, 1, 1,-1,-1, 1 /      ･･･ 符号検査用
C
C     (1) LIMIT CHECK
      READ(5,1000)    COUNT,(VALS(I),I=1,COUNT)
      WRITE(6,2000)   '(1) ',COUNT
      WRITE(6,2010)   (VALS(I),I=1,COUNT)
C
      DO 100 I = 1,COUNT
          ERRLIN(I) = '    '
  100 CONTINUE
      WRITE(6,2020)
C
      DO 110 I = 1,COUNT
          IF ( VALS(I) .GE. VLIMIT(I) ) THEN
              ERRLIN(I) = ' **'
          ENDIF
  110 CONTINUE
      WRITE(6,2030) ERRLIN
C
```

```
C
C     (2) SIGN CHECK
          READ(5,1000)     COUNT,(VALS(I),I=1,COUNT)
          WRITE(6,2000)    '(2) ',COUNT
          WRITE(6,2010)    (VALS(I),I=1,COUNT)
C
          DO 200 I = 1,COUNT
              ERRLIN(I) = '     '
   200    CONTINUE
          WRITE(6,2020)
C
          DO 210 I = 1,COUNT
              IF ( VALS(I) .GT. 0 .AND. VSIGN(I) .LT. 0     .OR.
                   VALS(I) .LT. 0 .AND. VSIGN(I) .GT. 0          ) THEN
                  ERRLIN(I) = ' $$ '
              ENDIF
   210    CONTINUE
          WRITE(6,2030)    ERRLIN
C
C     (3) PARITY CHECK
          READ(5,1100)    COUNT,(VALS(I),I=1,COUNT),TAIL
          WRITE(6,2100)   '(3) PARITY VALUE IS '
C
          ITOTAL = 0
          DO 310 I = 1,COUNT
              ITOTAL = ITOTAL + VALS(I)
   310    CONTINUE
          IF ( MOD(ITOTAL,2) .EQ. TAIL ) THEN
              WRITE(6,2110)
          ELSE
              WRITE(6,2120) MOD(ITOTAL,2)
          ENDIF
C
C     (4) TOTAL CHECK
          READ(5,1100)    COUNT,(VALS(I),I=1,COUNT),TAIL
          WRITE(6,2100)   '(4) TOTAL  VALUE IS '
C
          ITOTAL = 0
          DO 410 I = 1,COUNT
              ITOTAL = ITOTAL + VALS(I)
   410    CONTINUE
          IF ( ITOTAL .EQ. TAIL ) THEN
              WRITE(6,2110)
          ELSE
              WRITE(6,2120) ITOTAL
          ENDIF
C
          STOP
C
  1000    FORMAT(11I4 )
  1100    FORMAT(12I4)
  2000    FORMAT(' ',A4,'COUNTS OF DATA = ',I2,' :  ')
  2010    FORMAT('+',29X,10I4)
  2020    FORMAT(' ')
  2030    FORMAT('+',30X,10A4)
  2100    FORMAT(' ',A20)
  2110    FORMAT('+',20X,'RIGHT.')
  2120    FORMAT('+',20X,'WRONG. CORRECT VALUE IS < ',I4,' >')
          END
```

上記プログラムにおいて、(1)の限界値チェックと(2)の符号チェック処理はよく似ており、また、(3)のパリティーチェックと(4)の合計値チェック処理がやはり似ていることが読み取れる。

［ＦＯＲＴＲＡＮバージョン］ － SUBプログラムを使って処理するケース

まず、限界値チェック、および符号チェックを行う処理をサブルーチン化し、
次に、パリティチェック、および合計値チェックを行う処理をサブルーチン化する。
限界値及び符号検査用テーブルはメインプログラムにあるものとする。

```
      INTEGER*4   VLIMIT(10),VSIGN(10)
C
      DATA        VLIMIT/40,90,60,20,80,70,50,10,30,100/      ･･･ 限界値検査用
     1            VSIGN /-1, 1, 1,-1, 1, 1, 1,-1,-1, 1 /      ･･･ 符号検査用
C
C   (1) LIMIT CHECK
      CALL  CHECK1(1,'(1) ',VLIMIT)    ････    限界値チェック処理
C
C   (2) SIGN  CHECK
      CALL  CHECK1(2,'(2) ',VSIGN )    ････    合計値チェック処理
C
C   (3) PARITY CHECK
      CALL  CHECK2(1,'(3) PARITY VALUE IS')  ････   パリティチェック処理
C
C   (4) TOTAL  CHECK
      CALL  CHECK2(2,'(4) TOTAL  VALUE IS')  ････   合計値処理
C
      STOP
C
      END


    SUBROUTINE     CHECK1(C,TITLE,D)
      INTEGER      C
      CHARACTER*4  TITLE
      INTEGER*4    D(10)
C
      INTEGER*4    COUNT,VALS(10)
      CHARACTER*4  ERRLIN(10)
C
C
      READ(5,1000)          COUNT,(VALS(I),I=1,COUNT)
      WRITE(6,2000)         TITLE,COUNT
      WRITE(6,2010)         (VALS(I),I=1,COUNT)
C
      DO 100 I = 1,COUNT
        ERRLIN(I) = '    '
  100 CONTINUE
C
      WRITE(6,2020)
C
      DO 110 I = 1,COUNT
        IF( C .EQ. 1 ) THEN
C   (1) LIMIT CHECK
          IF ( VALS(I) .GE. D(I) ) THEN
            ERRLIN(I) = ' ** '
          ENDIF
        ELSE
C   (2) SIGN  CHECK
          IF ( VALS(I) .GT. 0 .AND. D(I) .LT. 0  .OR.
     1         VALS(I) .LT. 0 .AND. D(I) .GT. 0      ) THEN
            ERRLIN(I) = ' $$ '
          ENDIF
        ENDIF
  110 CONTINUE
      WRITE(6,2030) ERRLIN
C
      RETURN
```

```
C
 1000   FORMAT(11I4)
 2000   FORMAT(' ',A4,'COUNTS OF DATA = ',I2,' : ')
 2010   FORMAT('+',29X,10I4
 2020   FORMAT(' ')
 2100   FORMAT('+',30X,10A4)
        END



      SUBROUTINE     CHECK2(C,TITLE)
        INTEGER      C
        CHARACTER*4  TITLE
C
        INTEGER*4    COUNT,VALS(10),TAIL
C
        READ(5,1000)     COUNT,(VALS(I),I=1,COUNT),TAIL
        WRITE(6,2000)    TITLE
C
        ITOTAL = 0
        DO 100 I = 1,COUNT
            ITOTAL = ITOTAL + VALS(I)
  100   CONTINUE
C
        IF ( C .EQ. 1 ) THEN
C   (1) PARITY CHECK
            IF ( MOD(ITOTAL,2) .EQ. TAIL ) THEN
                WRITE(6,1010)
            ELSE
                WRITE(6,1020) MOD(ITOTAL,2)
            ENDIF
        ELSE
C   (2) TOTAL CHECK
            IF ( ITOTAL .EQ. TAIL ) THEN
                WRITE(6,1010)
            ELSE
                WRITE(6,1020) ITOTAL
            ENDIF
        ENDIF
C
        RETURN
C
 1000   FORMAT(12I4)
 1010   FORMAT('+',20X,' RIGHT.')
 1020   FORMAT('+',20X,' WRONG. CORRECT VALUE IS < ',I4,' >')
 2000   FORMAT(' ',A20)
C
        END
```

⑤　開いたサブルーチンと閉じたサブルーチン

今まで述べてきたサブルーチンを閉じたサブルーチンと呼び、それ自身がコンパイルの一つの単位となり、原始プログラムの形式としては、プログラムの開始（実行の入り口点となる）文と終了文（実行の終了とコンパイルの終了とがある）とが必ず存在する。

閉じたサブルーチンは、他のルーチンから呼び出されるときにはパラメータが渡される。これを引数といい、呼び側と呼ばれ側とで対応がついていなければならない。この引数並びのことを一般にソフトウェアではコーリングインタフェースという。

このインタフェースに準じてサブルーチンをコールすれば、サブルーチンは色々なプログラムから利用され、プログラムの再利用に役立ち、プログラム開発の生産性が向上する。

一方、開いたサブルーチンとは、色々なプログラムでよく使われるような特定の処理手続きを、コンパイル単位として閉じるのではなく、そのサブルーチンの名称をプログラム中で引用すると、コンパイル時に他のファイルから読み込まれ、原始プログラムの一部となる性格をもつ処理ルーチンである。

このルーチンは一般に、原始プログラムライブラリに登録されており、コンパイルの必要な時点でインクルードする方法をとっている。これに相当するものとして、ＣＯＢＯＬのＣＯＰＹルーチンや、アセンブラ言語のマクロ等がある。

３．モジュール分割と結合

前節で若干述べたように、ルーチン分けを行うこと自身をモジュール分けと考えて差し支えない。この考え方により高水準言語でモジュール分けを行うとすれば、
・ＦＯＲＴＲＡＮであれば、ＳＵＢＲＯＵＴＩＮＥ、ＦＵＮＣＴＩＯＮ
・ＰＬ／Ｉであれば、ＰＲＯＣＥＤＵＲＥ
・ＣＯＢＯＬであれば、ＳＵＢＲＯＵＴＩＮＥ
・Ｃであれば、ｆｕｎｃｔｉｏｎ
単位にプログラムを書くことになる。

このモジュール分けを行う場合留意すべきことは、
・閉じたサブルーチンであり、
・どのようなプログラムからも呼び出すことができ、
・単独でコンパイルできる
ことである。これにより、モジュールとしての独立性が高くなり、モジュール分けをきっちり行うことにより、モジュール自身の単体のテストが容易にできるばかりでなく、結合したときの全体のテストもやりやすくなることで、ひいてはプログラムの生産性向上に貢献することになる。

このモジュール分けを行うに当たって、好ましい分割を行うための考え方、また分割に際しての大きさに関する一応の目安を定めておくことが大切である。

- 分割のための考え方
  - テストが容易に行えること
  - プログラムの解読が容易であること
  - プログラムの保守性が高いこと
  - 機能拡張が容易であり、拡張性が高いこと
  - 再利用可能性が高いこと（分割したら再利用が利くとは限らない）
- モジュールの大きさが適切であること
  管理の面などからも、１モジュールが数十ステップでできていることが望ましい
- インタフェースの適切な量
  個々のモジュールの機能が大きすぎないこと、呼出手続きが楽であることが望まれる

なお、モジュールの独立性の高さは、以下の３点により決まってくる。

○　モジュールの機能が簡潔明快で、把握し易いこと
○　モジュール間の結合度が低い（詳細後述）
○　呼び出すときの、受け渡し情報量が少なく、かつ明確であること

ここで、モジュール間の結合度とモジュールの把握のし易さについて簡単に触れておこう。

①　モジュール間の結合性

互いに相手のモジュールについて知るべき情報は最小限であることが好ましく、モジュール間での情報の交換性が低いほど好ましい。ここでいう結合とは、直接的に呼び出し合うモジュールだけでなく、どのようなモジュール間でも情報を知り合えない方がよい。

・他のモジュールを呼び出す場合、情報はパラメータだけで受け渡すこと
・相手のモジュールの内部的な動きを知ったような制御パラメータは用いないこと
・共通アクセスがあるような領域、大域変数を用いた情報の受け渡しをしないこと
・他のモジュールの内部情報を自モジュール内で直接使わないこと

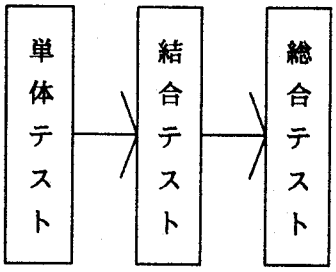
②　モジュール機能の把握の容易性

モジュールの機能は、何を提供するかを知らせるだけでよく、プログラムの内容についてまで呼ばれる側が公開する必要はない。上位モジュールに直接従属する下位モジュールは、上位モジュールの分かりやすさを確保するためにも、機能が簡潔に把握される必要がある。

・モジュールは機能を果たすだけでよく、利用者は呼び出すときの受け渡し情報だけ気をつければ良い
・モジュールの使用側は、モジュール内部でのデータの表現法や変化の具合を知る必要はない。単にデータを情報として渡せば良い

4．モジュール結合プログラムのテスト

モジュール分割されたプログラムは、単体テスト、結合テスト、総合テストの順でモジュールが結合され、徐々に全体的なテストに移る。特にモジュールが階層構造的に分割されている場合には、各段階にふさわしいテスト技法があるのでそれを解説する必要がある。



・単体テスト　－　モジュール毎のテスト。このテストのためにドライバ、スタブ（後述）が使われる。また、階層構造モジュールではトップダウンテスト、ボトムアップテスト方式がある。

・結合テスト　－　モジュールを結合しインタフェースを確認

・総合テスト　－　全モジュールを結合し、総合的なテストを行う。機能、性能、運用性など全てに亘る

● トップダウンテスト

上位モジュールから順に下位方向へとテストを進める。上位モジュールが下位モジュールを呼び出す必要がある場合、テスト未完了のモジュールに替えて、“スタブ”と呼ばれる一時的に作った仮のモジュールを使い代用する。例えば、最終的には必要となる入出力装置がまだ未納入のためその入出力ができないとき、あたかもそれをおこなったかのようなモジュールを作る。

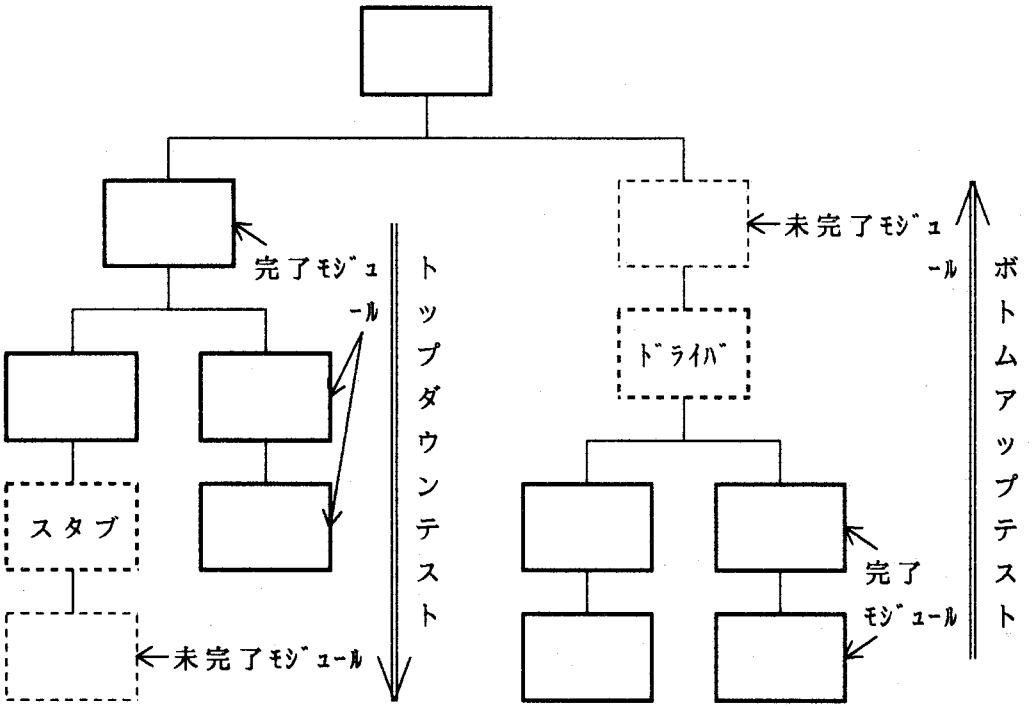
● ボトムアップテスト

下位モジュールから順に上位方向へとテストを進める。下位モジュールを呼び出す上位モジュールがテスト未完了である場合、それに替えて“ドライバ”と呼ばれる一時的に作った仮のモジュールを使い代用する。このドライバは下位モジュールを呼び出すインタフェースを作り出す必要がある。

単体テストでは、機能、およびインタフェースを正しく実現しているかを中心に行う。結合テストは、いくつかのまとまりがあり、単体テストを完了したモジュールをつなげてテストするもので、主としてモジュール間のデータの授受の正しさを確認する。

総合テストでは、システム全体での入力処理・計算処理・出力処理（機能面）の正しさを確認するとともに、求められる性能や応答性、操作性、さらには運用性についての評価なども同時にテストしなければならない。

≪トップダウンテストとボトムアップテスト≫



指導上の留意点

この章の内容については、ソフトウェアの入門者にとっては比較的大きなプログラムを相手にすることになる。モジュール分割の意義、プログラムテストの方法について理論的に深入りし過ぎることは禁物である。

適切な方法としては、長いプログラムを書かずに、サブルーチン化を押し進めることを常に強調し、生徒のプログラミング例を挙げて、サブルーチンの効能を解説することが好ましい。

また、良いプログラム分割の根拠についても、プログラム事例で説明することが肝要である。

用　語

目視チェック、デバッグ文、デバッグライト、デバッガ、静的解析用ツール、
コードオーディタ、
メインルーチン、サブルーチン、
モジュール分割、モジュール結合、
単体テスト、結合テスト、総合テスト、
トップダウンテスト、ボトムアップテスト、スタブ、ドライバ

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など。

○　プログラム技法に関すること
　　プログラムを作るための効果的な手法など。

○　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５章　アプリケーションに関連する事項

指導目標

アプリケーションの重要性について認識を高め、ソフトウェアの必要性から、ソフトウェアをどのように調達（新規開発か、外部からの購入かなど）するかを考えさせる。

アプリケーションパッケージの代表的な事例について触れる機会があると授業の効果を高めることもできる。

内容のあらまし

内　　容	説　明	議　論	机上実習	計算機実習
アプリケーションソフトウェア	基本ソフトウェア開発支援ソフトウェアとの区別をする。定義をはっきりさせておく。	知っているアプリケーションを挙げさせ、どんな機能を提供しているか説明させる。	特になし	特になし
アプリケーションの活用	ｱﾌﾟﾘｹｰｼｮﾝﾌﾟﾛｸﾞﾗﾑの調達方法、利用方法、アプリケーションのメリットを理解させる。	アプリケーションを開発するか、購入するかの判断をどうするか議論させる	特になし	特になし
アプリケーションの種類と活用上の留意点	アプリケーションプログラム事例と役割、使用上の留意点について理解させる	自分がアプリケーションを開発するとしたら、どんなことに重点を置くか考えさせる。	パッケージのパンフレットから、それについて机上評価をしてみる	汎用コンピュータ ＰＣ，ＷＳ等 ・情報検索 ・ＣＡＩ ・図形処理等

指導内容

1．アプリケーションソフトウェア

汎用コンピュータをメーカから購入する場合、大多数において、標準的なソフトウェアとして、オペレーティングシステム、各種ユーティリティプログラムをも併せて導入することになる。これにより、少なくとも利用者はプログラムを比較的容易に開発し、またそれを実際の業務遂行上のツールとして利用することとなる。

つまり、コンピュータメーカはその企業が製品化して販売を行っているコンピュータ及びその周辺機器類に対して、梱包を解き、必要なシステム生成を行いさえすれば利用者はすぐにでもそれらを大きな困難を伴うことなく使い始めることができる。メーカはコンピュータ資源を利用するための最小限のソフトウェアをインフラとして提供してくれるわけである。

これらを基本ソフトウェアと呼んでおり（詳細は第６章に記述）、一般の利用者（システム運用要員以外の情報処理技術者あるいは情報システムの利用者）にとっては、コンピュータと一体となった“もの”と捉えられる傾向もある。

ところで、企業、公益団体、学校などでいわゆる業務システムとよばれるものは、とりもなおさず、コンピュータ（マシンとＯＳ）とアプリケーションプログラムとから構成される。このプログラムは、通常はメーカから標準的に提供を受けるものではなく、利用者が業務目的に適する形で自ら開発するソフトウェアである。

アプリケーションは業務を効率よく処理するために作られるソフトウェアであり、当然ながら業務の違いによって異なるアプリケーションが開発されることになる。また、業務内容が類似していても、業種が異なる、あるいは企業風土が異なるなどにより、同じような利用目的を持ったソフトウェアであっても、それぞれ違ったシステムが出来上がるところにアプリケーションの価値が発生するとみることもできる。そういう意味でアプリケーションプログラムは、応用プログラム、あるいは適用業務プログラムとも呼ばれる。

アプリケーションプログラムには、統計処理、数値計算処理、シミュレーション、図形処理を必要とするようなデータ解析プログラム、在庫管理プログラム、給与計算プログラム等がある。

ここで、プログラムがどう作られ、どこに存在するかを整理しよう。

＜メーカが開発、提供＞

① オペレーティングシステム
② ﾕｰﾃｨﾘﾃｨ、ﾌﾟﾛｸﾞﾗﾐﾝｸﾞ支援ｿﾌﾄｳｪｱ
③ 適用業務ﾌﾟﾛｸﾞﾗﾑ開発用支援ｿﾌﾄｳｪｱ（ﾃﾞｰﾀﾍﾞｰｽ、ﾃﾞｰﾀ通信支援など）
④ ｿﾌﾄｳｪｱ・ﾊﾟｯｹｰｼﾞ（汎用的な作りではなく、ﾒｰｶ製品のみで利用可能）

＜ユーザ開発、利用＞

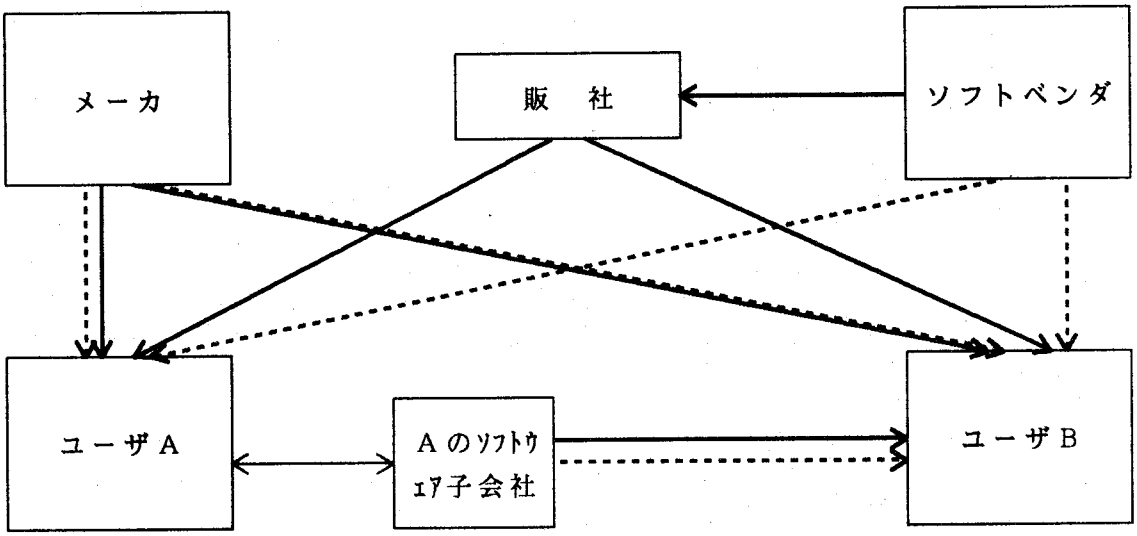
自社向け適用業務プログラム。メーカ、ｿﾌﾄｳｪｱ開発協力会社の技術支援を受けて開発及び利用。

＜独立ベンダ開発、提供＞

ソフトウェア開発支援ツールパッケージ
適用業務プログラムパッケージ

次に、これらのプログラムの流通について整理する。

- メーカは、ユーザに対してアプリケーションプログラムの販売とサポートを行う。
- ソフトベンダはソフトウェア販社に販売を依頼する。一方で、障害に対するサポートを実施する。（自社で販売を兼ねる場合も多い）
- あるユーザＡの情報処理専門子会社は、Ａ社向けに開発した適用業務プログラムを汎用化して、他のユーザに販売すると共にサポートも行う。



（注）⟶ システムの提供
⇢ サポート

2．アプリケーションの活用

アプリケーションプログラムの活用形態としては、以下の３種類に分けられる。

① 全面的な自社開発

開発資金一切を自社で負担し、自社の情報処理技術者を中心に設計から開発までを行う。運用、保守についても同様である。開発コストがかかりすぎたり、開発納期が守れない等の多くのリスクを伴うが、自社に最適な情報システムを作ることができる利点もある。

② 流通パッケージの利用

独立ソフトウェアベンダが作ったソフトを自社の業務処理用に購入して適用する。基幹事務処理系のアプリケーションパッケージの場合まだまだ市販ソフトウェアの利用が多いとはいえないが、技術計算系、ＣＡＤ／ＣＡＭ等エンジニアリング業務系にはかなり利用されている。

③ パッケージソフトのカスタマイズ

独立ソフトウェアベンダが作ったソフトを導入すると共に、自社の業務処理方式に馴染むように、あるいは特別に欲しい機能を追加することにより独自の利用を行う。

流通パッケージを利用するメリットとして、以下のような点が挙げられる。

- 開発コストが不要もしくは低コストで済む
- 購入ソフトウェアが優良であれば、業務遂行効率も上がり、色々な面で品質の向上が図れる
- 運用、保守、教育など外部に依頼することができ、情報化に対する余剰投資を避けられる

以下で、自社開発とパッケージソフトの流用についての比較を簡単に整理する。

	利　　点	欠　　点
自社開発	○ 自社に最適なアプリケーションを開発できる。保守も自社で可能	○ 開発に時間、コストがかかる ○ 保守に追われ新規開発が遅れる ○ 先進技術の導入が容易でない
パッケージ利用	○ 自社で開発できない優良なプログラムが手にはいる ○ 開発コスト、情報要員を削減可 ○ 広範な普及により、企業間でのデータ処理の互換性がでてくる	○ 自社の風土、手法に馴染むとは限らない ○ サポートがなければ不安である ○ 特殊な環境でしか動作しないと、拡張性に欠ける

3．アプリケーションの種類と活用上の留意点

以下でアプリケーションプログラムの例を挙げる。

共通応用プログラム	業務・業種別応用プログラム
・時系列分析 ・多変量解析 ・統計解析 ・線形計画 ・意思決定支援システム ・予測シミュレーション ・工程管理 ・情報検索 ・図形処理 ・ＣＡＤ／ＣＡＭ	・会計処理 ・人事管理 ・生産管理 ・在庫管理 ・受発注管理 ・プロセス制御 ・科学技術計算 ・ＣＡＩ／ＣＭＩ ・医療情報処理 ・構造解析

（出典：初級情報処理技術者育成指針：（財）日本情報処理開発協会・中央情報研究所）

例えば、統計解析アプリケーションプログラムには、統計データ処理に関する種々の計算機能を備えており、コマンドの呼出により一連の処理を操作できるように作られている。コンピュータの専門家でなくても比較的容易に機能を使いこなせる利点がある。

これらのアプリケーションを選択するときに留意すべき事項について以下に列挙する。

○　誤りが少ないこと
○　求めている精度が保証されること
○　操作性が容易なこと
○　誤った使い方に対する警告がでること
○　必要な主メモリ量や補助記憶の量が説明されていること
○　正確なマニュアルが整備されていること
○　どのような環境で利用できるか明確であること、かつ種々の環境で利用できること
○　適切なバージョンアップがあること

指導上の留意点

口頭ではなかなかアプリケーションプログラムの効能を説明することは容易ではない。コンピュータショー、データショー、ソフトウェアショーなどでアプリケーションのパンフレットや説明書をいくつか入手し、個々に説明をする方が具体的な話ができる。生徒にとっても、その方が楽である。

用　語

アプリケーションプログラム、応用プログラム、適用業務プログラム、
ソフトウェアパッケージ、アプリケーションパッケージ

第2種情報処理技術者試験

⑥　アプリケーションに関すること。
　　各種アプリケーションプログラムの使用目的、使用方法、用語など。

第６章　基本ソフトウェアに関連する事項

指導目標

基本ソフトウェアは、利用者にとってコンピュータの持つハードウェア的な諸機能を使いやすく、効率的に使用でき、さらに複数の利用者が同時に色々な処理を行う場合、それらの動作を円滑に進むようコントロールする仕掛であることを理解させる。

基本ソフトウェアが通常、コンピュータメーカにより標準的にハードウェアとセットで提供されることが多いため、利用者にとってあまり関心が持たれない領域ではあるが、実際には非常に重要なソフトウェアであり、コンピュータの利用価値を高める最重要な要素の一つであることを認識させる。

基本ソフトウェアは、従来はオペレーティングシステム、コンパイラ、リンカー、各種ユーティリティソフトウェアからなると指導されており、それは今日でも基本的には変わりはないが、近年はコンピュータを使いやすいユーザ環境までもが、その一部として取り上げられてもきている事から、パソコンや、ワークステーションを用いて、その辺についても実際には指導内容に含めてもよかろう。

基本ソフトウェアに関する学習の内、オペレーティングシステムについてはコンピュータ上でのプログラムの動作原理を理論的に、その他の部分についてはできるだけコンピュータ実習と合わせて理解させると効果が上がる。

本章に関して重点的に学習、習得させたい事項は以下の通りである。

①．コンピュータ利用者にとっての基本ソフトウェアの役割は何か
②．基本ソフトウェアは階層構造的に構成されること、各層毎に役割を有すること
③．ハードウェア資源の利用、プログラム動作環境のコントロール、ソフトウェア資産の管理等に関する機能を理解すること
④．オペレーティングシステムの機能、重要性につき原理的に理解すること
⑤．言語プロセッサ、モジュールの編集ユーティリティ、ライブラリ管理ユーティリティ等について機能を、コンピュータ実習過程で理解すること
⑥．商用コンピュータのスケール分類により、基本ソフトウェアのサービスする内容が異なることを知ること
⑦．ＯＳについては、汎用コンピュータ用、小型機やＷＳ用に使われるＵＮＩＸ、パソコンに流通している［ＭＳ－ＤＯＳ，ＯＳ／２，Ｗｉｎｄｏｗｓ］等について代表的なものを選択して、その事例引用により使い方が異なることも認識する

内容のあらまし

内　　容	説　明	議　論	机上実習	計算機実習
基本ソフトウェアについて	基本ソフトウェアの役割、守備範囲、階層構造につき理解させる。	汎用コンピュータ、ＷＳ、ＰＣの各基本ソフトウェアの特徴や違いにつき議論	基本ソフトウェアの構造図を紹介し、思想を理解させる。	特になし
ハードウェアとソフトウェアの関係	管理すべきハードウェア資源と、ＯＳから見た制御方法につき理解させる。	ＯＳがないとユーザはどんな繁雑なことをさせられるか議論。	特になし	特になし
オペレーティングシステム	オペレーティングシステムのトータルな目的、実現構成につき理解させる。	汎用コンピュータ、ＷＳ、ＰＣの基本ソフトの重点の置き方の相違につき議論	２種試験問題のオペレーティングシステムに関連する問題	特になし
オペレーティングシステムのメカニズム	多重ﾌﾟﾛｸﾞﾗﾐﾝｸﾞジョブ制御、ジョブ、タスク、データ、記憶管理の方法につき理解させる。	ＲＡＳＩＳの確保のためにＯＳはどのような工夫を行っているか議論	多重プログラミングの原理、タスクスケジューリングに関する理解度確認	特になし
言語に関連すること	言語の種類と特徴を捉える。よく使われる言語に関する知識を習得させる。	コンピュータを理解するのにアセンブラと高水準言語いずれが適するか議論	特になし	汎用ｺﾝﾋﾟｭｰﾀ ＰＣ，ＷＳ
サービスプログラム	連携編集、ローダ、分類・併合デバッグ支援、ライブラリ管理	ｻｰﾋﾞｽﾌﾟﾛｸﾞﾗﾑなしではﾕｰｻﾞはどんな繁雑なことをさせられるか	２種試験問題のサービスプログラムに関連する問題	汎用ｺﾝﾋﾟｭｰﾀ ＰＣ，ＷＳ

内　　容	説　明	議　論	机上実習	計算機実習
ユーティリティプログラム	データセットに関する操作をどのように行うかを理解させる。	特になし	特になし	汎用ｺﾝﾋﾟｭｰﾀ ＰＣ，ＷＳ

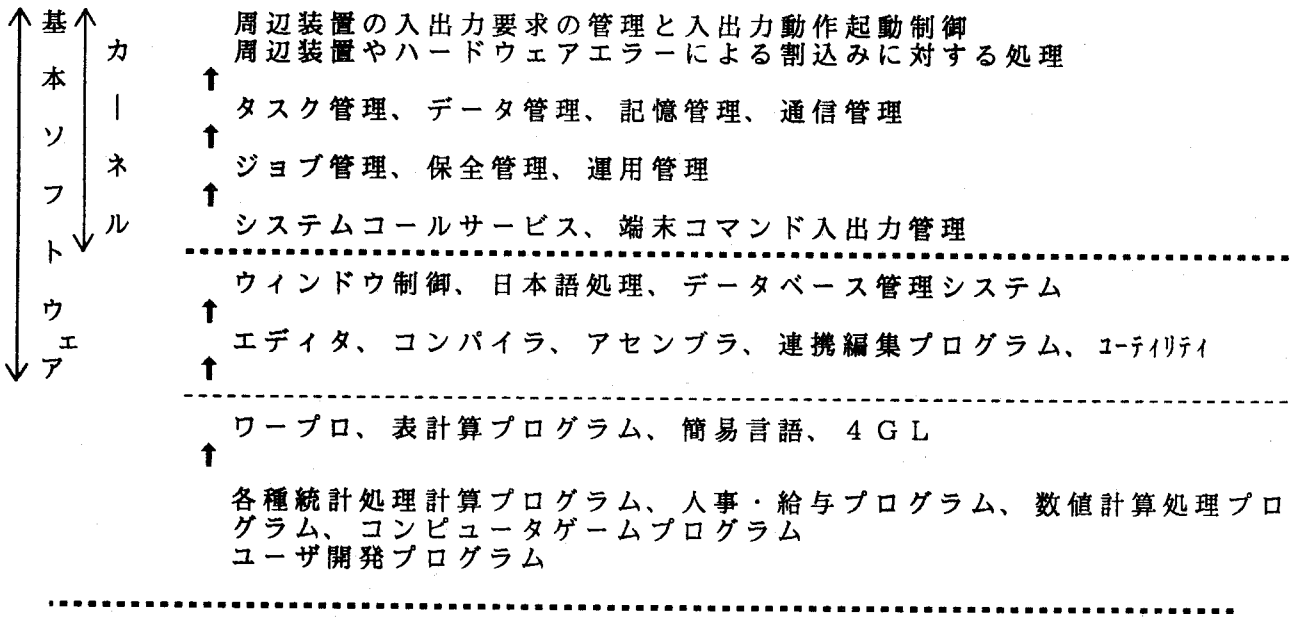
指導内容

1．基本ソフトウェアについて

ハードウェアから利用者がコンピュータと向かい合う最初の接点となる部分までの種々のプログラムやデータ、情報、利用上、管理上のルールを含めた一切を基本ソフトウェアと呼ぶこととする。狭義には、オペレーティングシステム、言語プロセッサ、連携編集プログラム、標準ライブラリ等を指す。

（１）基本ソフトウェアの階層構造

ハードウェアの制御に近い部分から、ユーザプログラムの管理・制御、ユーザによるコンピュータ操作の制御に相当する部分迄の役割や、ソフトウェアとしての実現構造から階層的に捉える事ができる。



（２）基本ソフトウェアの捉え方と役割

基本ソフトウェアを簡単に捉えると、以下の３つの部分に分けることができる。

①　オペレーティングシステムの中核部

バッチ処理、オンライン処理、タイムシェアリング処理、個人コンピュータ処理など、コンピュータ処理の主目的の違いにより、オペレーティングシステムの提供する機能、カバーする役割の範囲が異なる。

基本的には、コンピュータの複数人による共同同時利用に対する、
・資源の効率的な、円滑な利用のための割当て
・単位時間当たりの処理量の増大
・コンピュータ資産の安全な管理、運用
・コンピュータに記憶されたプログラムの適正な実行計画と、実行順序、実行状態の管理についてコントロールするものである。

また、この部分はユーザからは自分のプログラムにより直接データをアクセスしたり、プログラム動作の制御を渡したりすることはできない部分である。特別のモードで実行される。

②　コンピュータ操作支援に関わる部分

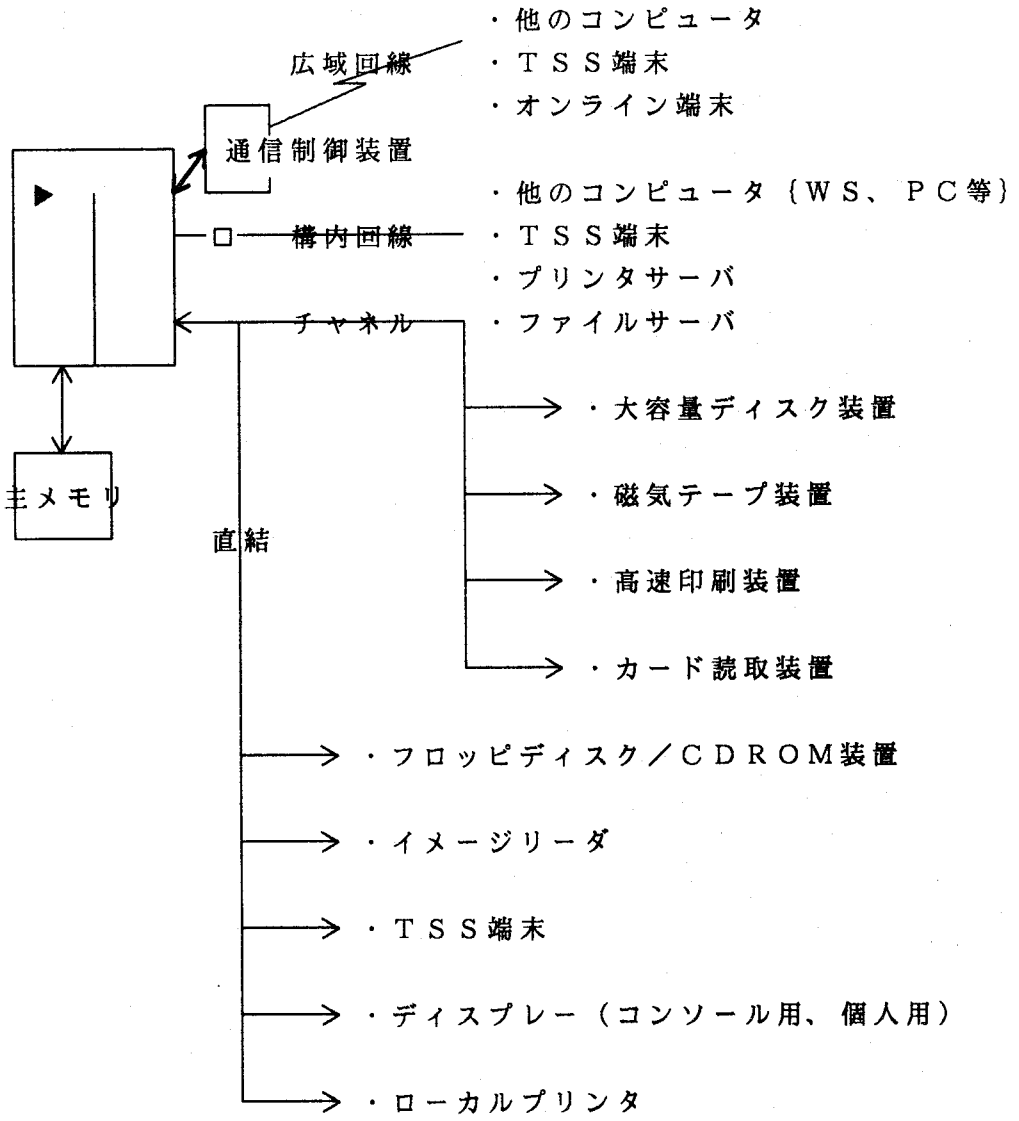
最近のコンピュータでは個人の利用環境の高度化を図るため、マシンの種類によらない統一されたインタフェースにより、日本語の処理やマルチウィンドウ機能、グラフィックユーザインタフェースが提供されている。この機能はオペレーティングシステムの中核とは切り離されており、マンマシンインタフェース技術の向上によりますます発展を遂げている。

③　プログラム開発用の道具となる部分

プログラムの作成、実行モジュールの作成、実行モジュールのテストなど、コンピュータ言語プログラミングによってソフトウェアを作る際の道具立てや、実行確認環境を提供する。汎用コンピュータにおけるＴＳＳ端末、個人利用としてのＷＳ，ＰＣディスプレー／キーボードから起動するコマンドなどもこの類である。

基本ソフトウェアは、コンピュータからみるとやはりプログラムであること、カーネル部よりも利用者に近い部分は、利用者が開発するプログラムと本質的には大きな違いはないことを認識しておく必要がある。

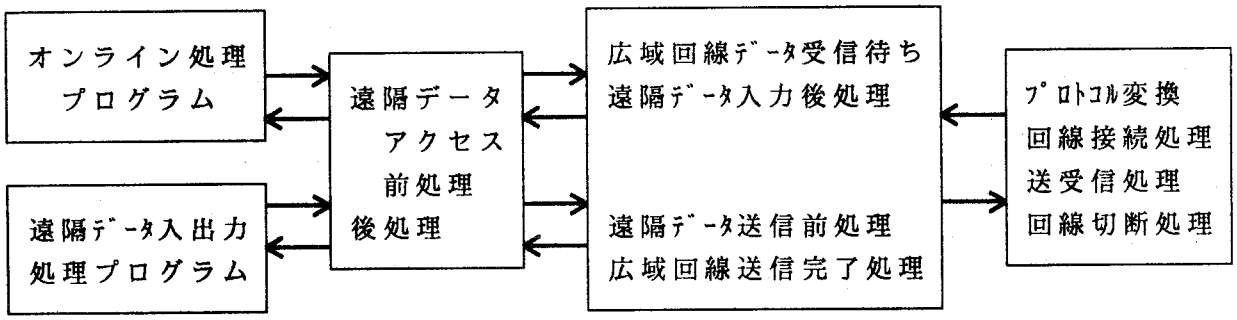
2．ハードウェアとソフトウェアの関係



コンピュータ利用者が開発するプログラムから使用できるハードウェアとしては、上図のように中央演算処理装置（ＣＰＵ）、主メモリ、広域回線通信用制御装置、構内回線入出力、コンピュータに直接つながる各種入出力装置があり、その他特殊な装置として音声装置、ディジタイザ等も扱える。

主メモリを除き、これらの周辺装置に対する入出力はオペレーティングシステムの機能として、入出力動作を各種制御装置に起動指令を出すデバイスドライバ、および制御装置からの入出力動作の完了通知後の処理を行う割込み処理ルーチンにより実行される。

（１）広域回線通信



① 遠隔データ受信

例えば、オンライン処理プログラムが遠隔データを受信したい場合、ＯＳに対して入力要求を出す。ＯＳではそのプログラムからの入力依頼を記録しその準備を行う。一方で、通信回線経由で通信制御装置にデータが届くとＦＥＰはＯＳのデータ形式に変換してそれを中央処理装置に送る。ＯＳはそれをオンライン処理プログラムが理解できる形式に変換してその受信を告げる。

② 遠隔データ送信

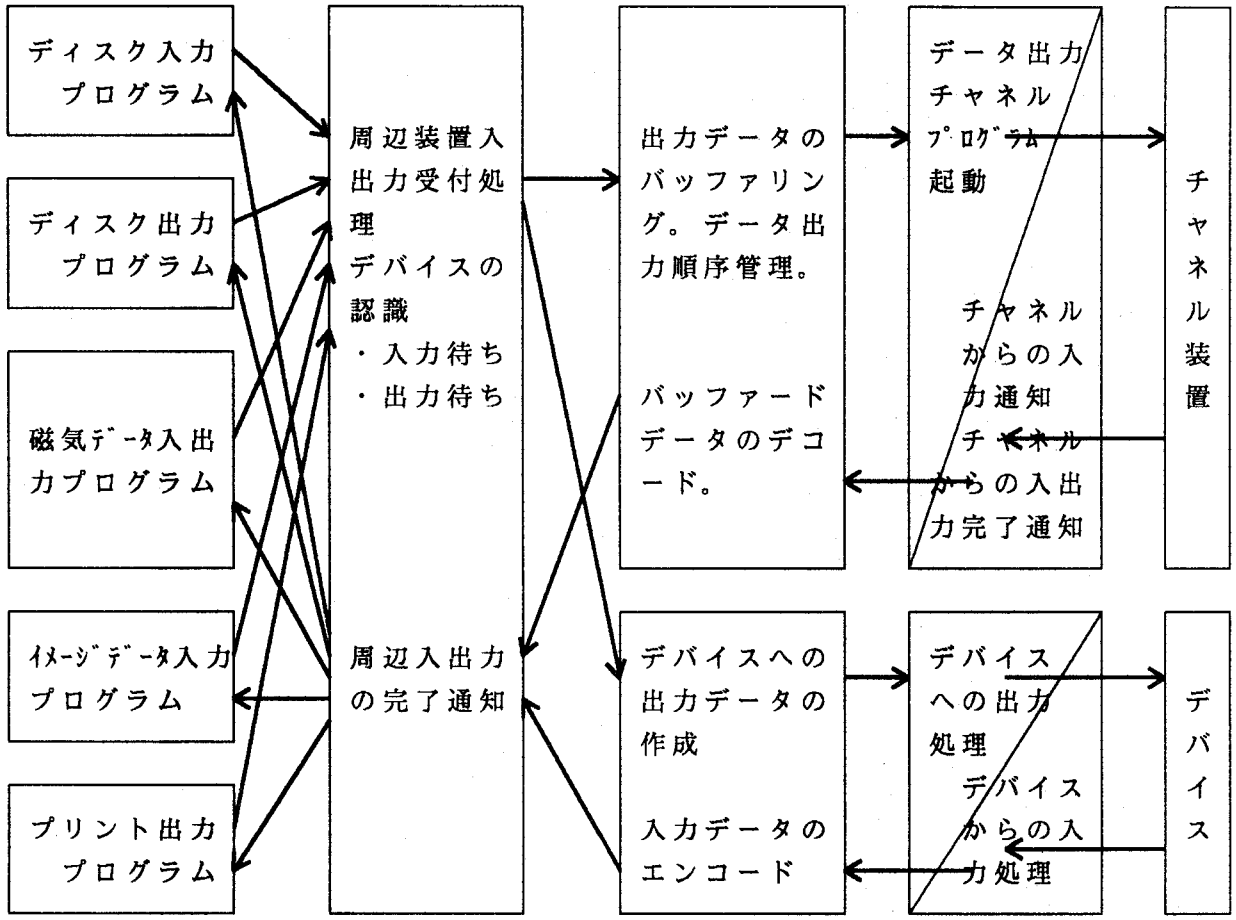
例えば、あるプログラムが遠隔データを送信したい場合、ＯＳに対して出力要求を出す。ＯＳではそのデータを回線への送信可能なデータに変換をし、ＦＥＰに対して回線を経由した遠隔への送信指令を出す。ＦＥＰは遠隔通信用のデータ形式に変換してそれを送信する。相手への送信が確認されたらその旨をＯＳに知らせると、さらにＯＳは送信完了をユーザプログラムに知らせる。

プログラムからは一見自らが遠隔データの送受信を行っているように見えるが、オペレーティングシステムのデバイス・インデペンデントな入出力方式機能の提供により利用者は特別な苦労をせずに回線送受信が行えるようになる。

また、オペレーティングシステムからみると、通信制御装置の通信プログラム（ＦＥＰ）により回線の種類、プロトコルの違いを意識しない回線送受信ができることになる。

なお、通信制御装置は通信入出力に関わるオーバヘッドに対してコンピュータの負荷を減らすために重要な要素であるが、小型コンピュータやＷＳ，ＰＣ等には通常このような装置は接続されない。小型系のマシンからも遠隔データ送受信要求はあるが、ゲートウェイサーバを介してそれを行う。ゲートウェイサーバは自ら通信制御装置の機能を行っている。

（２）周辺装置との入出力



ユーザプログラムからは、“ファイル名”、“デバイス名”等を指定しての装置独立な要求形式で入出力を要求してくる。

ＯＳは、出力要求に対しては、出力先デバイスの特定、チャネルや入出力制御装置へのデータ形式の変換、チャネルや制御装置への出力指令を作成して、出力要求を出し、出力の完了を待つ。

入力要求に対しては、チャネルや制御装置からの入力があったときに、プログラムが求める形式で入力領域にデータを移し、入力の旨を知らせる。

3．オペレーティングシステム

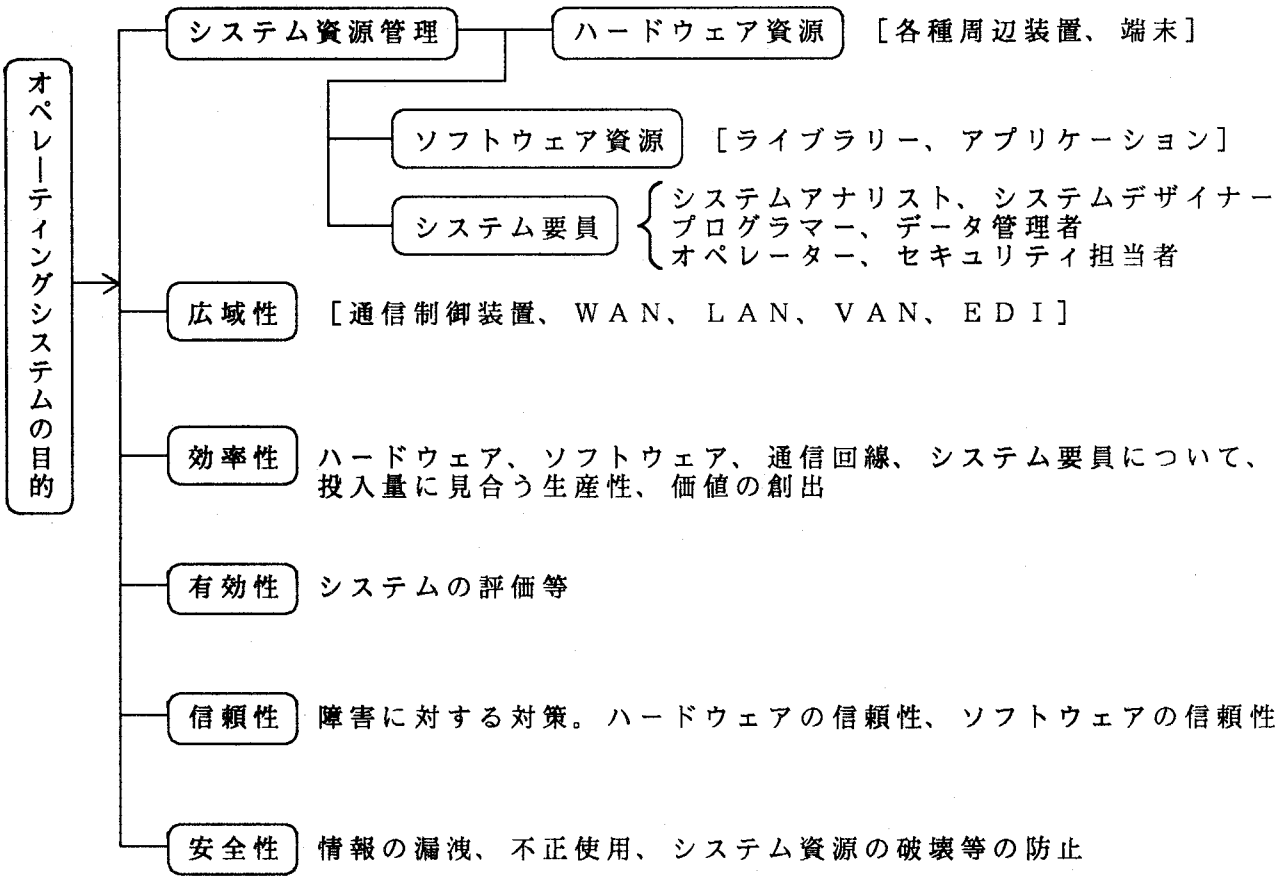
オペレーティングシステムは、ハードウェア資源の効率的な活用と稼働、コンピュータ利用者に対する使い勝手の良い環境、ソフトウェア資産の有効活用およびその安全な管理体系を提供してくれるシステムであることを理解させる。

オペレーティングシステムの働き、動作原理を正しく理解し、プログラムを作る場合にプログラムの特性に合わせて、あるいはプログラムの目標に合うように、提供される機能を適切に選択して活用できる能力が求められることを認識させる。

（1）オペレーティングシステムの目的

コンピュータの種類、スケールによって重要視される点は異なる。この部分は従来の参考書などでは汎用メインフレームにおけるオペレーティングシステムの目的が強調されることが多かったが、小型マシン、ＷＳ、ＰＣ系ではユーザ環境としての高機能性が注目されていることも合わせて指導する必要があろう。

①　汎用メインフレームＯＳの目的



(a) ハードウェア資源の有効活用
多重プログラミング環境におけるハードウェア資源の利用率の向上

(b) ハードウェア資源の管理
メモリや入出力装置の仮想化など

(c) 性能の確保
・スループットの向上（単位時間当たりのデータなどの処理量の増大）
ジョブの連続実行性の強化、多重度の向上
・ターンアラウンドタイムの短縮（ジョブの投入から結果の回収までの所要時間の短縮）
多重度の向上、ジョブスケジューリングの改善
・応答時間の短縮（ＴＳＳ、オンラインリアルタイム処理における作業の指示から結果の受取りまでの応答の短縮）

(d) 機密保護機能の確保
・オペレータの資格の確認
・ファイルやデータのアクセス権の確認
・端末からの接続（ログオン）資格の確認
・ファイル内容の暗号化
・通信回線データの暗号化

(e) プログラミング作業負荷の軽減
・コンパイラのプログラムソースコードの静的な誤りチェック機能
・汎用ライブラリの管理
・ロードモジュール生成機能の多様化
・デバッグ機能の改善

(f) 操作性の向上
マンマシンインタフェースの向上

(g) 運用性の向上
・ジョブの自動スケジューリング
・システムの障害の早期発見
・オンライントレース機能

(h) ＲＡＳＩＳの確保
信頼性（Reliability）、可用性（Availability）、保守性（Serviceability）、
完全性（Integlity）、　機密性（Security）

② 小型、ＷＳ、ＰＣ系ＯＳの目的

小型系のコンピュータでは、汎用コンピュータと比べ、いくつかの点で異なる要素を持っており、したがってそのＯＳの目的、機能、実現方法も変わってくる。

(a) コンピュータの構成でみると、全体的なスケールはずっと小さい（演算装置の速度、主メモリの容量、周辺入出力装置の容量、チャネルの構成、・・）。
(b) 計算処理量、求められる信頼性、安全性なども低い。
(c) 処理の多重度はずっと低く、また個人環境としての充実性が求められる。
(d) 一つのコンピュータが抱える資源、資産の量もはるかに小さい。
(e) 操作性の高さが重要視される。
(f) 対話型の利用環境であり、ソフトは殆ど対話により起動される。
(g) 個人間の情報交換の機会がずっと多い。

小型系のマシンでは、オペレーティングシステムに大きな負荷をかけず、周辺の基本ソフトウェアを充実させることによって使い勝手を重視している。安価で提供することも基本的な命題であり、ＣＰＵの演算装置の処理スピードはともかく、ハードウェアに大きなコストをかけないため、ＯＳも自然とシンプルになってくる。

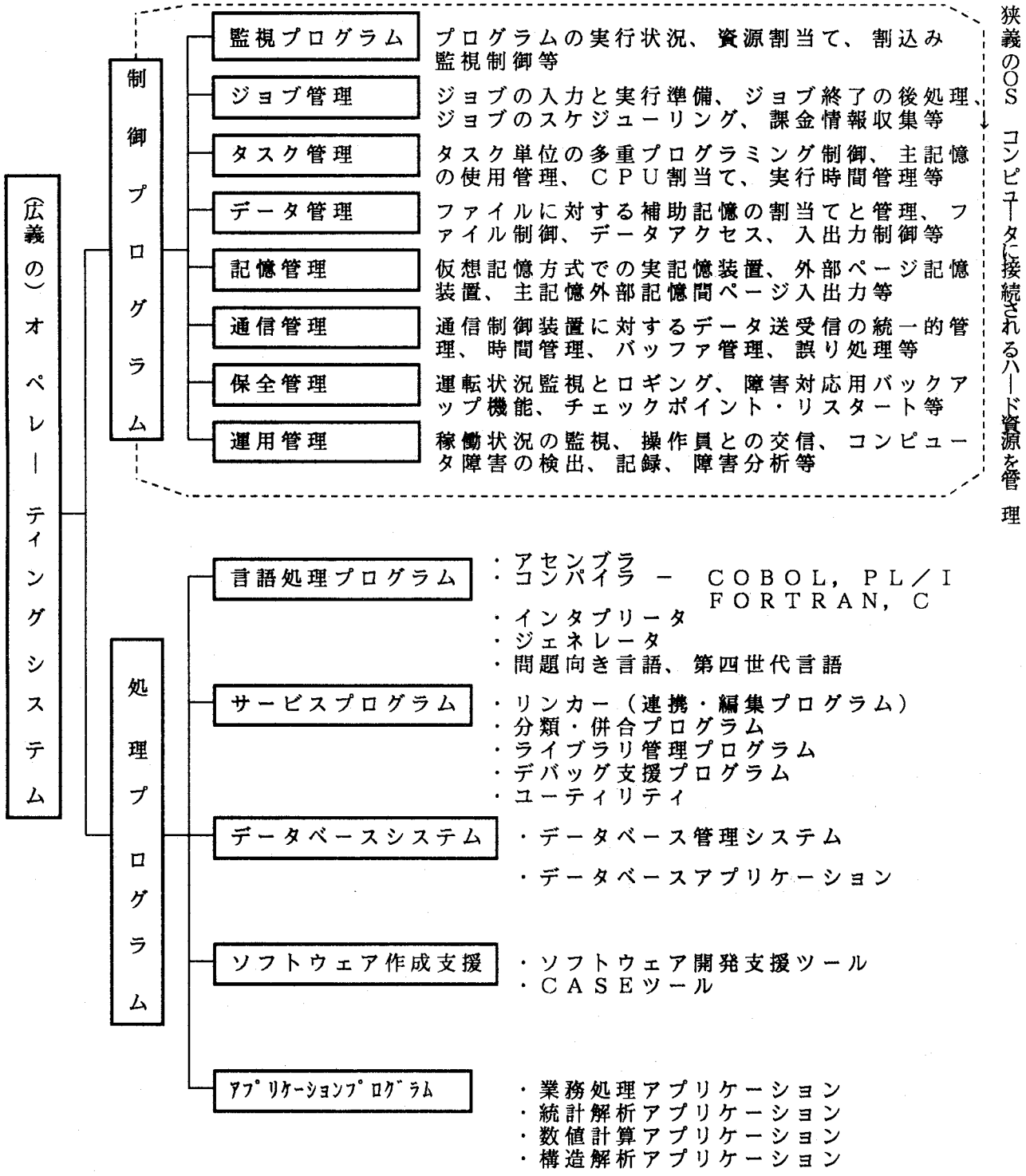
汎用コンピュータと大きく異なる部分は、ファイルの構造である。多重階層型の木構造をとっており、これにより豊富に提供されている各種のコマンド（ユーティリティ）の利用を非常に簡便にしている。

利用形態は、スタンドアローンな利用からネットワークを介した分散型の利用に大きく変化をしている。また大きな仕事はサーバマシンに、個人レベルの仕事をクライアントマシンで処理をするというクライアント／サーバ型のコンピュータ環境が主流になりつつある。

このような状況を反映して、小型系のオペレーティングシステムでは、カーネル部分は汎用コンピュータに比べ次のような特徴を有している。

・ＪＣＬがなく、ジョブという概念は殆どない
・ＴＳＳ型のサービスである
・性能よりも、コスト効果的な資源の使い方をする
・ネットワーク通信とそれにからむ処理が非常に重視される
・ディスプレイに関連し、ウィンドウを管理することが基本となっている
・ＧＵＩ（グラフィックユーザインタフェース）の高度化に力が注がれている
・マルチメディア入出力サポートが果たされつつある

（２）オペレーティングシステムの機能構成

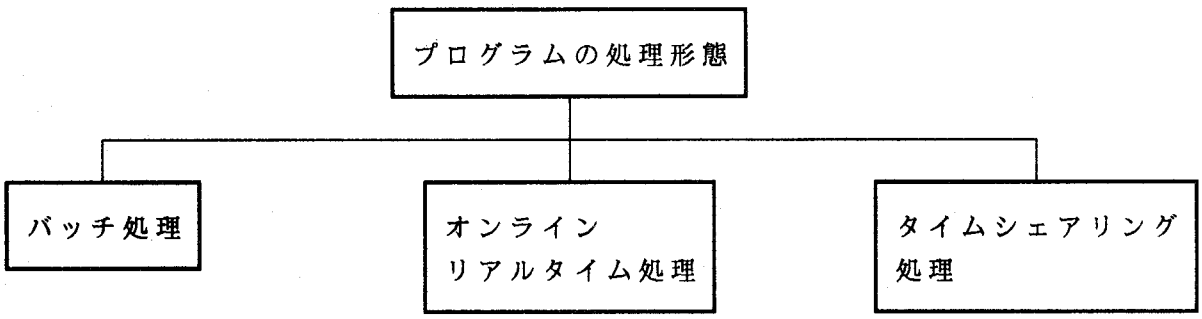


4．オペレーティングシステムのメカニズム

（1）プログラムの実行管理

オペレーティングシステム（制御プログラム）では、同時に要求される複数個の各種プログラムを時分割方式で処理をしなければならない。各プログラムはＯＳに対してどのような単位で、どのようなトリガーで処理要求を行うかを知る必要がある。

汎用コンピュータにおいては、利用者にとってのＯＳに仕事の実行を要求するインタフェースは、計算センターの窓口にジョブの実行依頼をし、操作員がそれをカード読み取り装置を介して投入する、端末からジョブファイルの実行要求指示を行う、端末からオンライン処理要求指示を行う等によるのが一般的である。

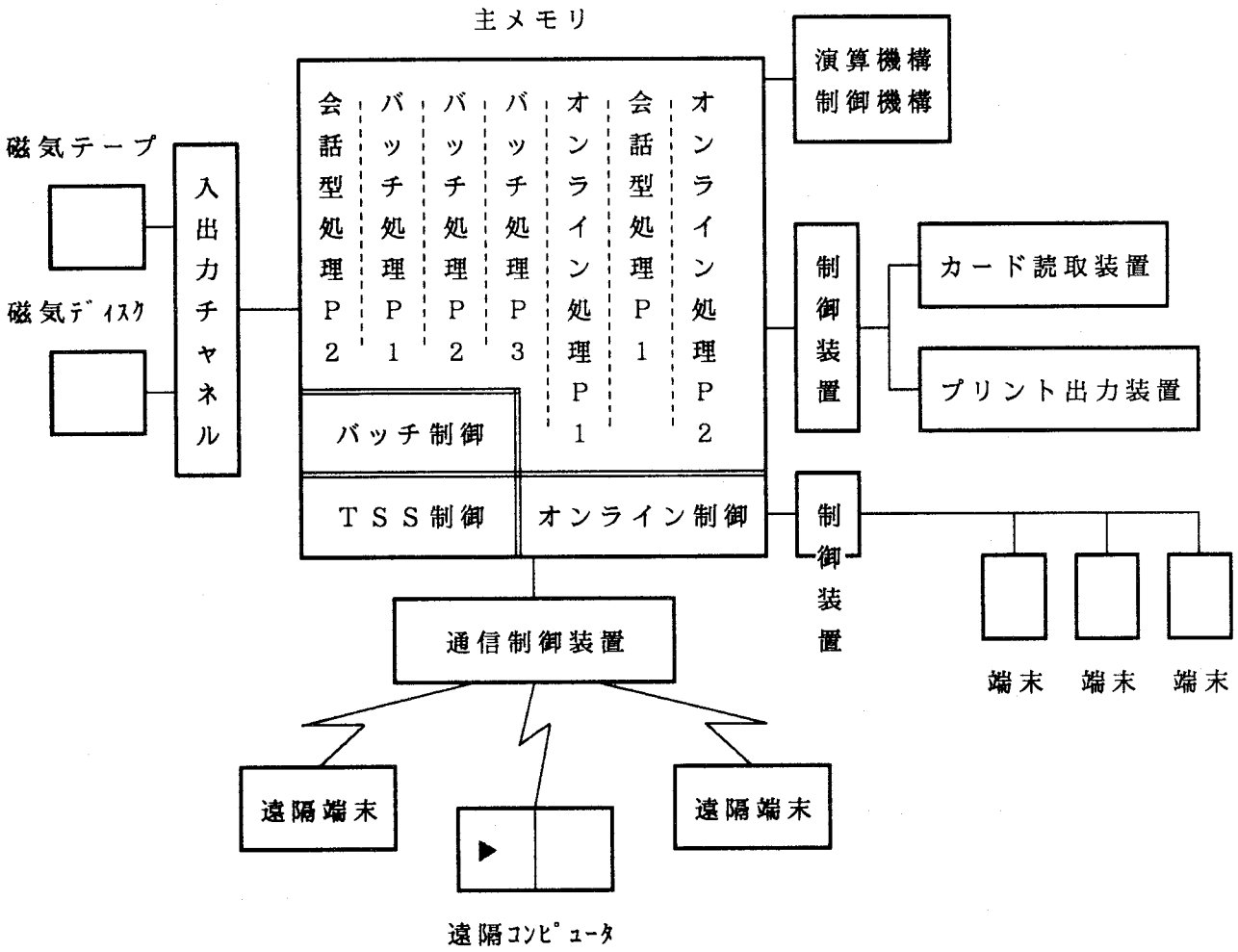


- “ジョブ”という概念を１単位として処理要求を出す。
  ジョブは、プログラムとデータおよびその他の処理要求を一まとめにしたものであり、通常操作員や遠隔地から回線端末を通してコンピュータに投入される。どのような仕事を要求するかは、“ジョブ制御言語”を用いて利用者が記述する。
  業務取引システムにおける月次処理、年次処理等大量のデータ処理、計算処理を行うのに適する処理形態である。

- 遠隔地にある端末からデータやコマンドを入力して、通信回線経由で中央のコンピュータと直接、データの照会、問い合わせ、更新等を行う。端末からの各種要求は即時処理が行われ、短時間で処理結果が返送され端末に出力される。
  ＪＲ、航空機の座席予約各種催し物チケット、旅館の予約処理等もこの処理に相当する。
  企業においては、在庫の照会、受注管理、生産管理等の業務もこの処理形態で行われている。

- 地理的に分散する複数の利用者が、コンピュータを同時に端末から使用する。コンピュータとの対話により作業が進められ、ターンアラウンドタイムが短縮される。プログラム開発や、量の多くない各種計算業務処理に適す処理形態である。
  ＴＳＳ端末からの“ログオン”、“ログオフ”は基本的にバッチ処理のジョブと同じ概念である。
  ＴＳＳ対話処理においても、ジョブ制御ファイルを作成してそれをコンピュータに投入できる。

≪多重プログラム実行処理≫



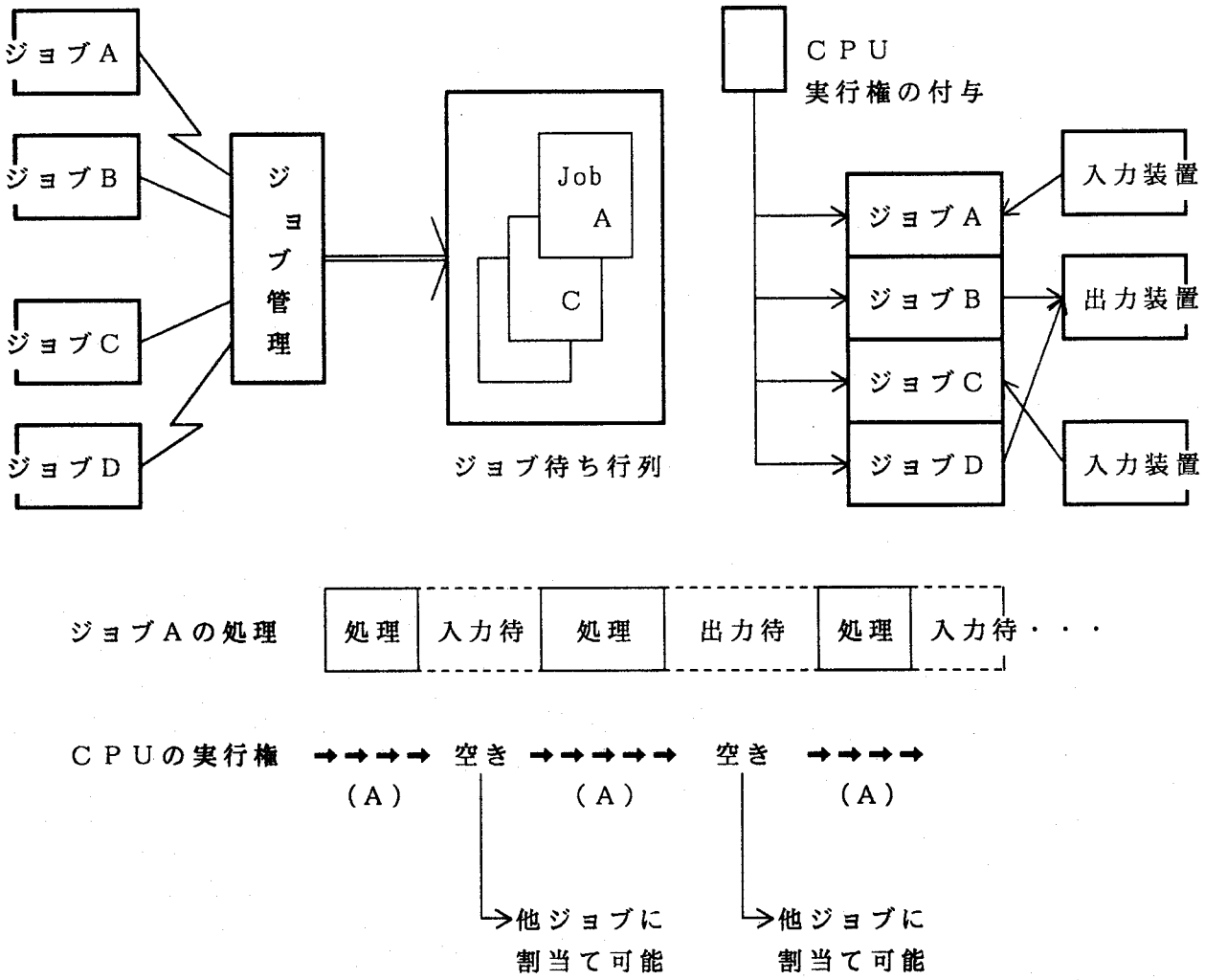
（２）多重プログラミング

処理形態の違い（バッチ、ＴＳＳ、オンラインリアルタイム）に関わりなくほとんどのプログラム処理では、必ず周辺装置に対する入出力処理を伴う。多重処理においてあるプログラムが実行制御権を持っているときに、入出力を実行した場合、データの転送速度はＣＰＵの処理速度に比較すると格段に遅い。したがって入出力の完了を待っている時間帯に、他のプログラムが実行制御権を獲得できれば非常に高価なＣＰＵの使用率は上がる。

このように、ＣＰＵの待ち時間ができるだけ発生しないよう実行待ちになっている各種のプログラムが空き時間を使うことにより、見た目には同時に複数個のプログラムが処理される仕掛を多重プログラミングと呼んでいる。

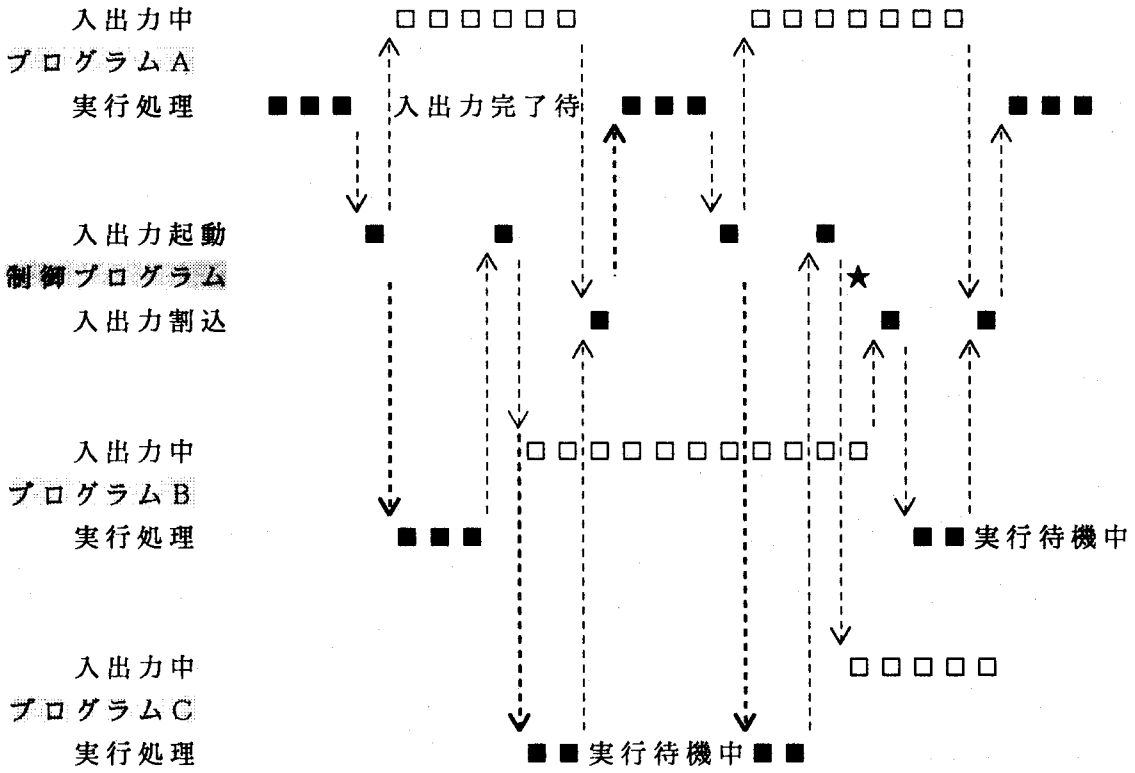
多重処理の実行制御の単位はＯＳの管理から捉えると厳密には“タスク”になる。この単位は、仕事の発生がバッチであろうと、ＴＳＳであろうと、あるいはオンラインリアルタイムであろうと共通の実行処理対象である。

しかし、ここでは説明上の混乱を避けるために、実行制御の単位をジョブとして考えることとする。



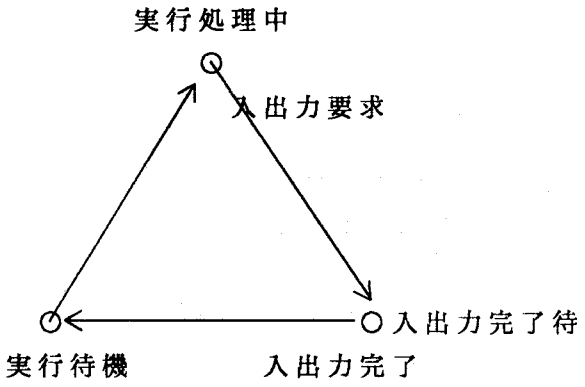
ＣＰＵの空き状態になったときは、他の実行待機中のジョブをジョブ待ち行列の中から探し、最も割当て権の高いジョブが“処理”を行うことになる。

≪多重プログラミングの原理≫



（注）★：ＣＰＵアイドル。　ジョブ実行優先順位はＡ，Ｂ，Ｃの順とする。
プログラムＡ～Ｃは、それぞれジョブＡ～Ｃに対応する処理対象であるとする。

≪ジョブの実行状態の遷移≫



（３）ジョブ制御

バッチ処理プログラム、ＴＳＳ処理プログラムはオペレーティングシステムのジョブ管理機能によりコントロールされ、実行が行われる。ジョブとはコンピュータに仕事を依頼する単位であり、ジョブ制御言語（ＪＣＬ：Ｊob Control Ｌanguage）で処理する内容を記述する。

ジョブ制御言語は、コンピュータの利用者が何をして欲しいかを実行あるいは指定したい順に記述する手段である。この言語は他のコンピュータ言語と同様、一連のジョブ制御ステートメントを書くことにより、一種のプログラムを形作る。これは後述のジョブ管理プログラムに対してのジョブ制御の情報となる。

商用計算機では、ジョブ制御言語は他のコンピュータ言語とは異なり、言語仕様はオペレーティングシステムによりまちまちである。また、ジョブ制御言語の種類により、

・ジョブ制御ステートメントの記述順に処理が進められる静的な記述法

・処理の実行順序を色々な条件によりに変えられる動的な記述法

がある。メインフレームのバッチ処理におけるジョブ制御ステートメントは静的な記述法が一般的である。

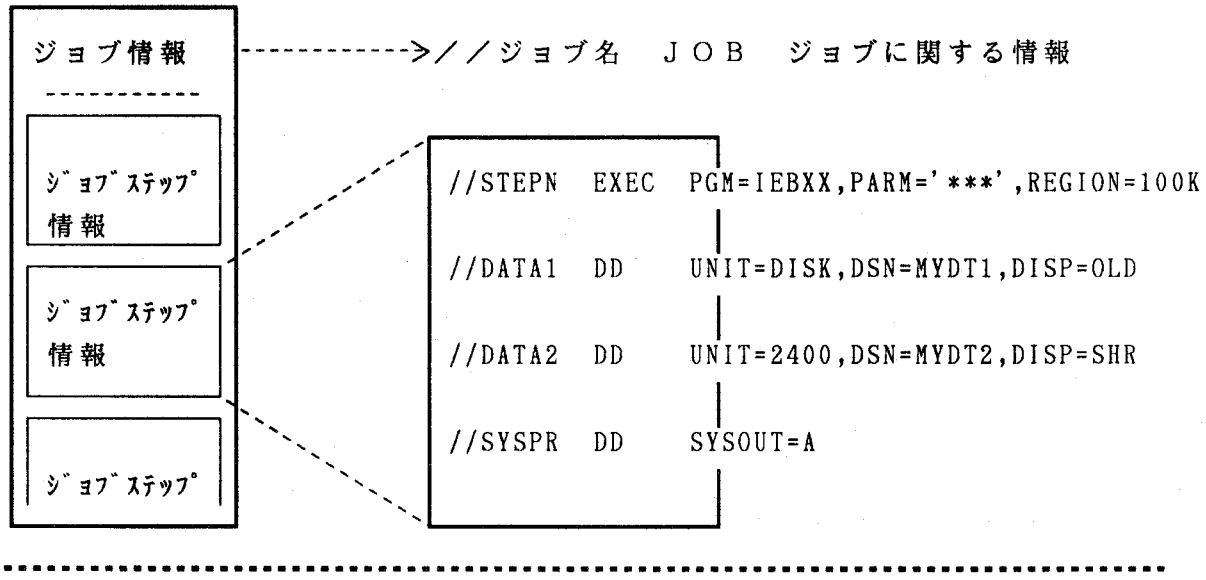
ジョブ制御言語の機能（ジョブ制御ステートメントで表す）は、以下に代表する内容を提供する。

◇ジョブの定義　　課金（アカウント名）先情報、制限時間等

◇プログラムの実行　　プログラム名、パラメータ、実行条件(メモリ量、制限時間)等

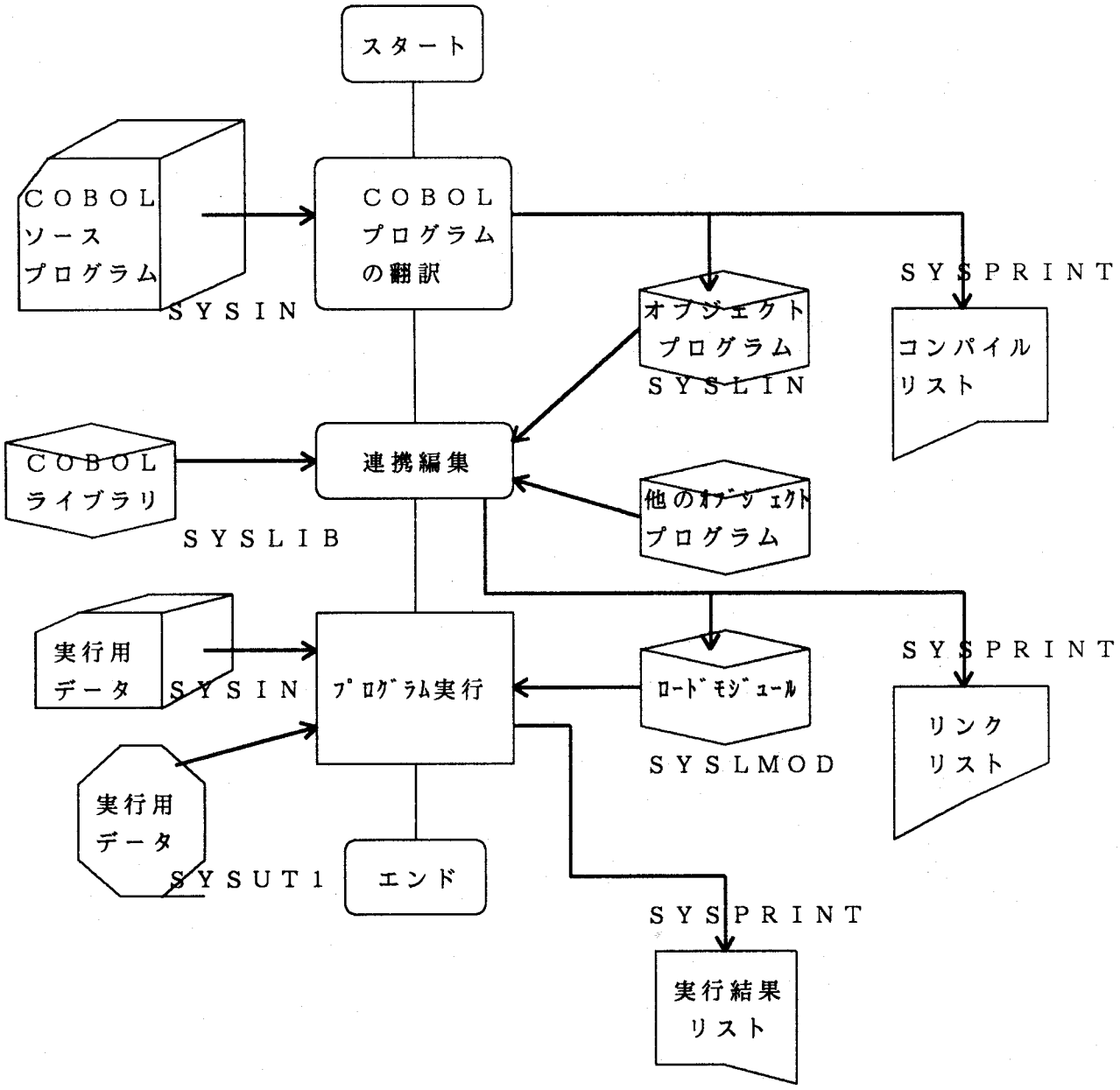
◇ファイル（データセット）の指定　　ファイル名、アクセス種類、レコード形式、ブロック化因数等

≪ジョブ制御ステートメントの書き方≫



≪ジョブ制御の流れとＪＣＬの関係≫

あるＣＯＢＯＬソースプログラムを翻訳（コンパイル）し、そのオブジェクトプログラムと既存の他のオブジェクトプログラムおよびＣＯＢＯＬライブラリとを連携編集しロードモジュールを作成する。できたロードモジュールを実行するという流れを考える。



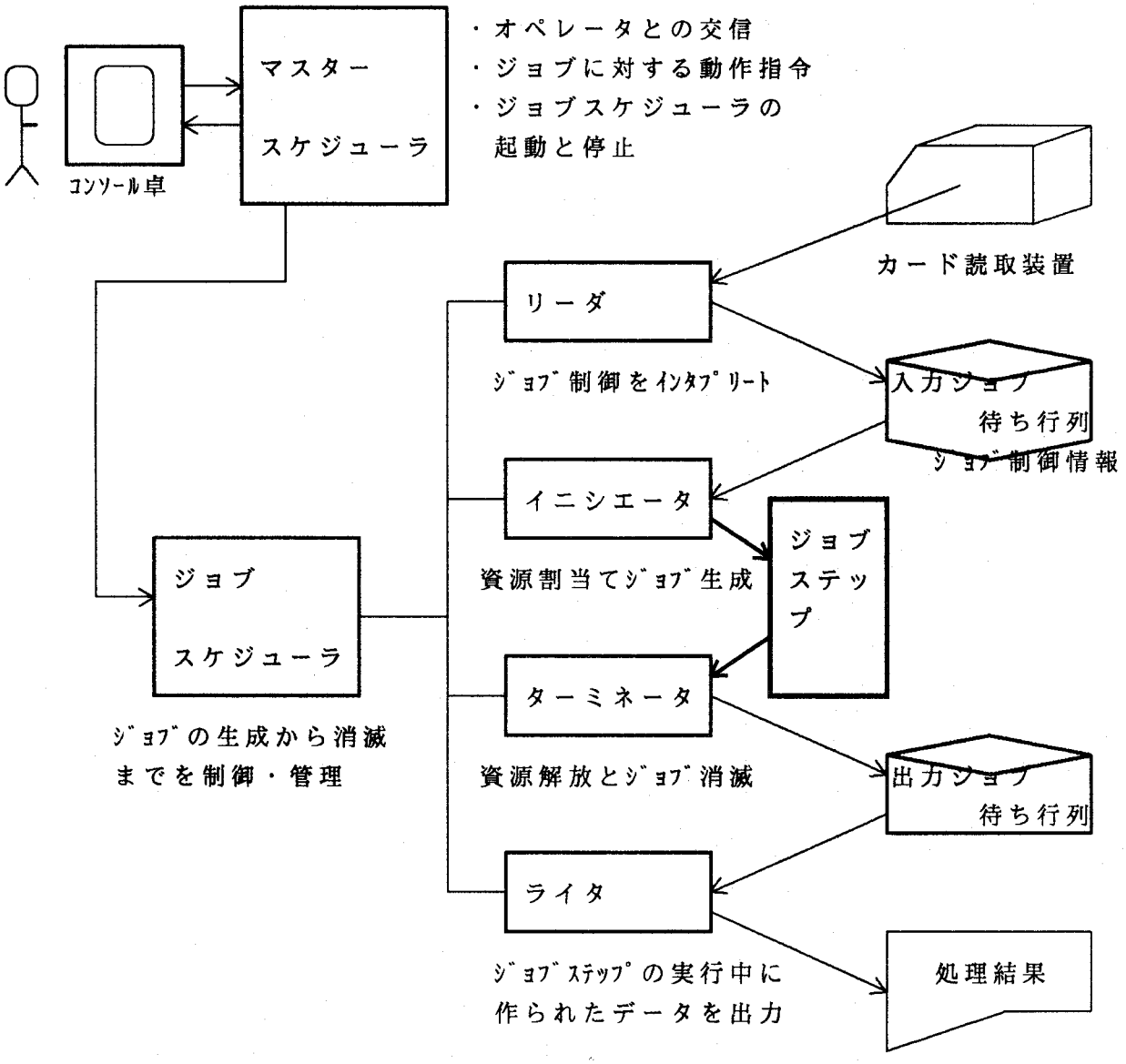
≪前頁の処理に対応するジョブ制御言語≫

```
//P930205  JOB   '....'                                          ジョブの定義
//COBC     EXEC  PGM=IKFCBL00,REGION=200K          ＣＯＢＯＬコンパイルの実行
//STEPLIB  DD    DSN=SYS2.LNKLIB,DISP=SHR
//SYSUT1   DD    DSN=DISK,SPACE=(460,(700,100))
//SYSUT2   DD    DSN=DISK,SPACE=(460,(700,100))   コンパイル時のワークディス
//SYSUT3   DD    DSN=DISK,SPACE=(460,(700,100))   クの割当て指定
//SYSUT4   DD    DSN=DISK,SPACE=(460,(700,100))
//SYSPRINT DD    SYSOUT=A
//SYSLIN   DD    DSN=&LOADSET,DISP=(MOD,PASS),UNIT=DISK,              X
//               SPACE=(80,(800,200)),DCB=(BLKSIZE=3200,LRECL=80)
//SYSABEND DD    SYSOUT=A                        コンパイル異常終了時のﾒｯｾｰｼﾞ
//SYSLIB   DD    UNIT=2314,DISP=SHR,DSN=P930205.COPYLIB,              X
//               VOL=SER=XKUNREN
//SYSIN    DD    *
  ＣＯＢＯＬ　ソースプログラムデック
/*
//COBL     EXEC  PGM=IEWL,PARM='LIST,XREF',COND=(5,LT,COBC)          連携編集
//SYSLIN   DD    DSN=&LOADSET,DISP=(OLD,PASS)
//         DD    UNIT=2314,DISP=SHR,DSN=P930205.MYOBJ,VOL=SER=XKUNKEN
//SYSLMOD  DD    DSN=&CUST,DISP=(MOD,PASS),UNIT
//SYSLIB   DD    DSN=SYS2.LINKLIB,DISP=SHR
//SYSUT1   DD    UNIT=(SYSDA,SEP=(SYSLIN,SYSLMOD)),                   X
//               SPACE=(1024,(50,30),,CONTIG)
//SYSPRINT DD    SYSOUT=A

//COBG     EXEC  PGM=*.COBL.SYSLMOD                                  プログラムの実行
//SYSIN    DD    *
  実行時にプログラムが入力するデータのデック
/*
//SYSUT1   DD    UNIT=2400,VOL=SER=XT930205,DSN=MTDT1,DISP=OLD
```

（４）ジョブ管理について

コンピュータ利用者が制御プログラムに対して仕事の依頼をする基本的な単位はジョブ制御言語で記述した“ジョブ”であり、制御プログラムはスループットを向上させるべく投入されたジョブの連続処理を行う。この仕事を司るのがジョブ管理プログラムであり、各ジョブに対してコンピュータ資源を効率的に割り付け、ジョブステップの流れを適切に管理する。

ジョブの管理は、制御プログラムとオペレータとのやりとり、ジョブ単位の管理、ジョブスッテプ単位の管理という、管理単位のレベルに応じ階層的に行われる。



ジョブは、リーダ、イニシエータ、ターミネータ、ライタにより一連の流れが制御・管理される。

① リーダ　　ジョブ入力装置から、ジョブ（カードデックや、コマンドファイルなど）ストリームを入力し、ＪＣＬの内容を解釈しながらジョブ制御情報を作って入力ジョブ待ち行列にキューする。同時に、実行処理時に入力されるデータについてはスプールデータセットに格納する。

② イニシエータ　　入力ジョブ待ち行列の中から優先順位の高いジョブを選択し、ジョブの実行を行う。取り出したジョブに対して、ジョブステップ毎に順に流れを制御する。各ジョブステップに対しては、入出力装置やデータセットの割当てを行いジョブステップタスクを生成する。

③ ターミネータ　　ジョブステップの実行が終了すると、割り当てられた資源を解放する。次に処理をすべきジョブステップが有れば再びイニシエータに制御がわたる。なければジョブを終了し、出力ジョブ待ち行列にキューする。

④ ライタ　　出力ジョブ待行列から処理結果の情報を読み取り出力装置に書き出す。出力情報としては、各ジョブステップの処理結果や制御プログラムが付加したが含まれる。

（５）タスク管理について

利用者がコンピュータに投入するジョブは、１つまたは複数個のプログラムを順次実行する一連の処理手続きにより構成される。ジョブ制御言語ではプログラムを実行する単位を“ジョブステップ”と呼び、利用者はプログラムの実行とそれに対するコンピュータ資源の割当てはこのジョブステップ単位に行う。

ジョブスケジューラは、イニシエータとターミネータを介してこのジョブステップに対応するプロセスに関する管理・制御を行うが、ジョブステップに対応するプログラム処理を実行する段階には、ＣＰＵの実行権を付与される単位である“タスク”を生成する。多重プログラミングの概念においてＣＰＵの実行権が与えられるのは制御プログラムの部分を除き、唯一このタスクという実行管理対象に対してである。

多重プログラミング環境において、タスクを生成し、ここのタスクを制御・管理する必要性は次の理由により生じる。

見かけ上の同時動作をする各タスクは、

・処理中に、必要時間数分ＣＰＵを使用し、必要量の主メモリを確保し、必要量の周辺装置を確保するとともに、処理終了後にはそれらを解放する
・タスクに設定されている優先順位によりＣＰＵの実行権を付与する
・実行処理中に種々の割込みがかかり、実行を中断してその割込み処理を優先的に行わなければならない

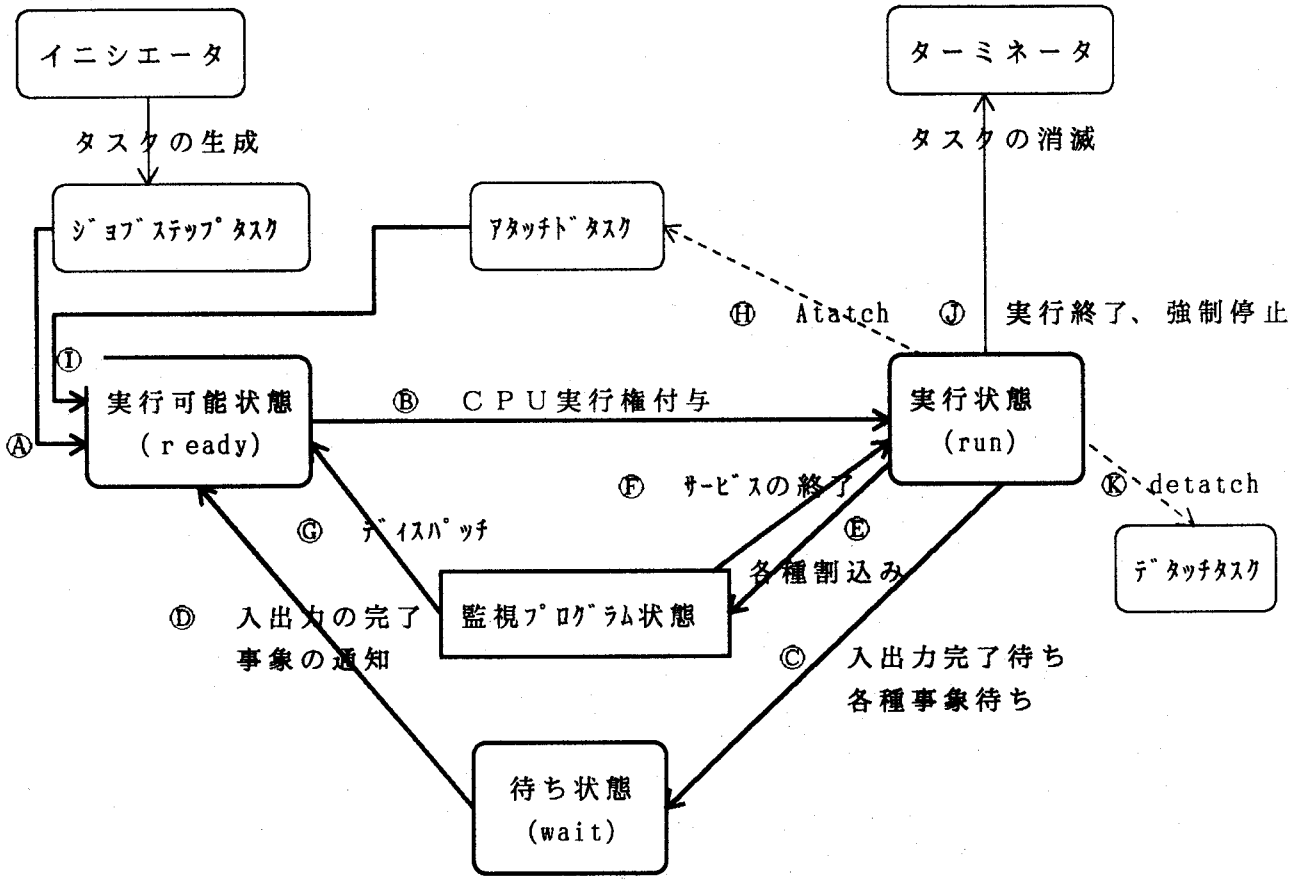
等により、実行状態の監視と資源割当て解放管理を受けなければならない。

以下で、タスク管理に必要とされる処理について説明する。

① タスクの状態管理

ジョブ管理のイニシエータによって、ジョブステップの実行開始に伴いタスクが生成されるとそれは実行処理が可能な状態となる。また、実行が正常に終了したり、あるいは異常な状態が発生し、強制的に終了しなければならなくなると、タスクは消滅し、ターミネータによってジョブステップの実行が終了させられる。

この間タスクは次のいずれかの状態を遷移し、その状態に対する管理が行われる。



Ⓐ イニシエータにより生成されたジョブステップタスクが実行可能状態となる

Ⓑ ディスパチングされたタスクが実行状態になる

Ⓒ 実行中のタスクが入出力完了または事象の発生を待つために待ち状態となる

Ⓓ 待ち状態のタスクに関連する入出力完了または事象が通知され、実行可能状態になる

Ⓔ タスクの実行中、各種割込みが入る。（システムサービスを要求する割込みもある）

Ⓕ 監視プログラムコールに対する処理を終え、要求した実行中のタスクに戻る

Ⓖ タスクスケジューラは実行可能状態キューから最優先のタスクに実行権を付与する

Ⓗ　実行中のタスクがサブタスクをatatchするとサブタスクが生成される
Ⓘ　atatchされたサブタスクが実行状態になる（実行可能キューにエントリーされる）
Ⓙ　実行中のタスクが終了するとターミネータがそのタスクを消滅する
Ⓚ　実行中のタスクがサブタスクをdetatchするとサブタスクが消滅する

② タスクのスケジューリング

実行可能状態にあるタスク（実行可能タスクキュー）の中から最優先のタスクを選択してＣＰＵの実行権を付与することをディスパッチングといい、これを行うのがタスクスケジューラ（ディスパッチャーともいう）。

どのタスクが最優先となるかの選択基準は制御プログラムにより異なる。

③ 割込み制御

実行中のタスクの処理を中断させ、割り込みの種類に対応する処理を優先して行う。この処理は制御プログラムの一部である。

(a) 内部割込み

◆機械割込み　　ハードウェアの誤動作、電源異常等が通知される
◆プログラム異常　　プログラムの実行中に不正が発生したことを通知される
◆監視プログラムコール　　利用者プログラムから制御プログラムに対して、各種のサービス要求が出された

(b) 外部割込み

◆入出力完了
◆入出力異常終了
◆オペレータからのコール
◆タイマ割込み

④ 主記憶域割当て

生成されたタスクが必要とするプログラム領域（プログラムの命令やデータを置くために必要な領域）を割り当てる。必要主メモリ量は、ロードモジュールから静的に計算され確保されるタイプと、オーバレイプログラムのように子セグメントをロードするときにメモリ量が計算されたり、プログラムが実行中にさらに主メモリの獲得を行うときのように動的に確保されるタイプとがある。

なお、最近のコンピュータではプログラムが要求するメモリは仮想記憶空間において確保される場所とサイズであり、主メモリを直接確保することを意味しない場合が一般的である。

（６）データ管理について

コンピュータ利用者が自らが物理的な入出力資源を制御・管理したり、複数の利用者により共有化されるべきデータやファイルを各人が管理することは非常に非効率なことである。

したがって、データやファイルが統一的に管理され、また入出力装置の物理的な特性に依らない利用方法が求められることからデータ管理が重要となってくる。その代表的な機能としては、名前によるＤＡＳＤ上のファイルの管理、媒体（入出力装置）に依存しないプログラムからの利用手続きの提供、高効率な入出力（物理的な入出力回数等の削減）、性能的にも、スペース量的にも効率の良い記憶領域の割付、ファイル破壊や不認可アクセスからの保護等がある。

以下に制御プログラムのデータ管理の機能の中で重要な要素につき簡単に説明する。

①　データセット制御（ファイル制御）

利用者がシンボル名を使って各種データの入出力を可能とする制御のことで、そのデータセット名と確保されたスペースに関する情報がボリュームラベルに記述される。

(a) データセットの識別

各データセットは自身のラベルにデータセット名を記録しており、利用者がジョブ制御ステートメント中にボリューム名を指定すると、ボリュームラベルから該当データセットを識別できる。

(b) カタログ管理

データセットの名前、所在場所をセットにした索引をカタログに登録する。データセット名が与えられたときにそれを探すためのものであり、このカタログのことをディレクトリともいう。

②　データセット（ファイル）のアクセス機能

入出力装置と主記憶装置との間でどのような方式でデータの転送を行うかを制御する機能。（これをアクセス方式と呼ぶ）

このアクセス方法に関連する概念としては、

・データセットの編成方法
・データの記録形式
・データのブロッキング

が代表的なものである。

③　入出力資源の割当て

利用者がプログラム言語を用いてデータセット入出力を行う場合、入力の対象として入出力装置名あるいはデータセット名を直接書くとすれば、プログラムを作る手続きとしては最も簡単であろう。

もしそのプログラムが一時的に利用される使い捨て的なものであれば、それも悪くはなかろう。小型系のマシンのオペレーティングシステムでは、しばしばこのようなスタイルがとられる。（ＪＣＬがないためもあろう）

ここで考慮すべきことは、プログラムが汎用的な位置づけにあったり、データの入出力媒体が状況により変化することを想定することである。プログラムにおける入出力ステートメントでは入出力装置からの独立した単位で入出力の要求を記述しておき、その単位がプログラムの外枠で入出力装置やデータセットとの対応関係がとられれば、プログラムを変更せずにいつでも入出力先を変えることができる利便性が得られる。

入出力装置の指定は、ジョブ制御言語のデータ定義ステートメントについて、装置の通番やデータセット名を与えることにより動的に行える。

また、ＪＣＬのデータ定義ステートメントにおいては、ディスク上にデータセットのスペースを確保するときに、その割当てをトラック単位、シリンダ単位に、さらに、連続したトラック、シリンダにとる指定も可能であり、これによりディスクアクセスの時間を短縮することも可能となる。

④　入出力制御システム

入出力装置との実際のデータ転送は、直接利用者プログラムが行うのではなく、制御プログラムの中の入出力制御システムによって処理される。この部分では機械語を用い、また通常の命令でなく、入出力制御専用の命令が使用されて、一連の装置入出力の手続きが実行される。

（７）記憶域管理

タスクに対して、その生成時あるいは実行処理中に必要な記憶域領域の割当てやその解放を行う必要がある。

多重プログラミング制御下では、主記憶域の管理は大変繁雑なものであるが、仮想記憶方式の実現によって、利用者（タスク）は、タスクに割り当てられた仮想記憶空間に対してのみのメモリの割当てを考慮すればよく、トランスペアレント（主記憶域の使用状態に依らない）なアドレス空間が提供されるという、非常に楽に大きな空間で仕事ができるようになった。

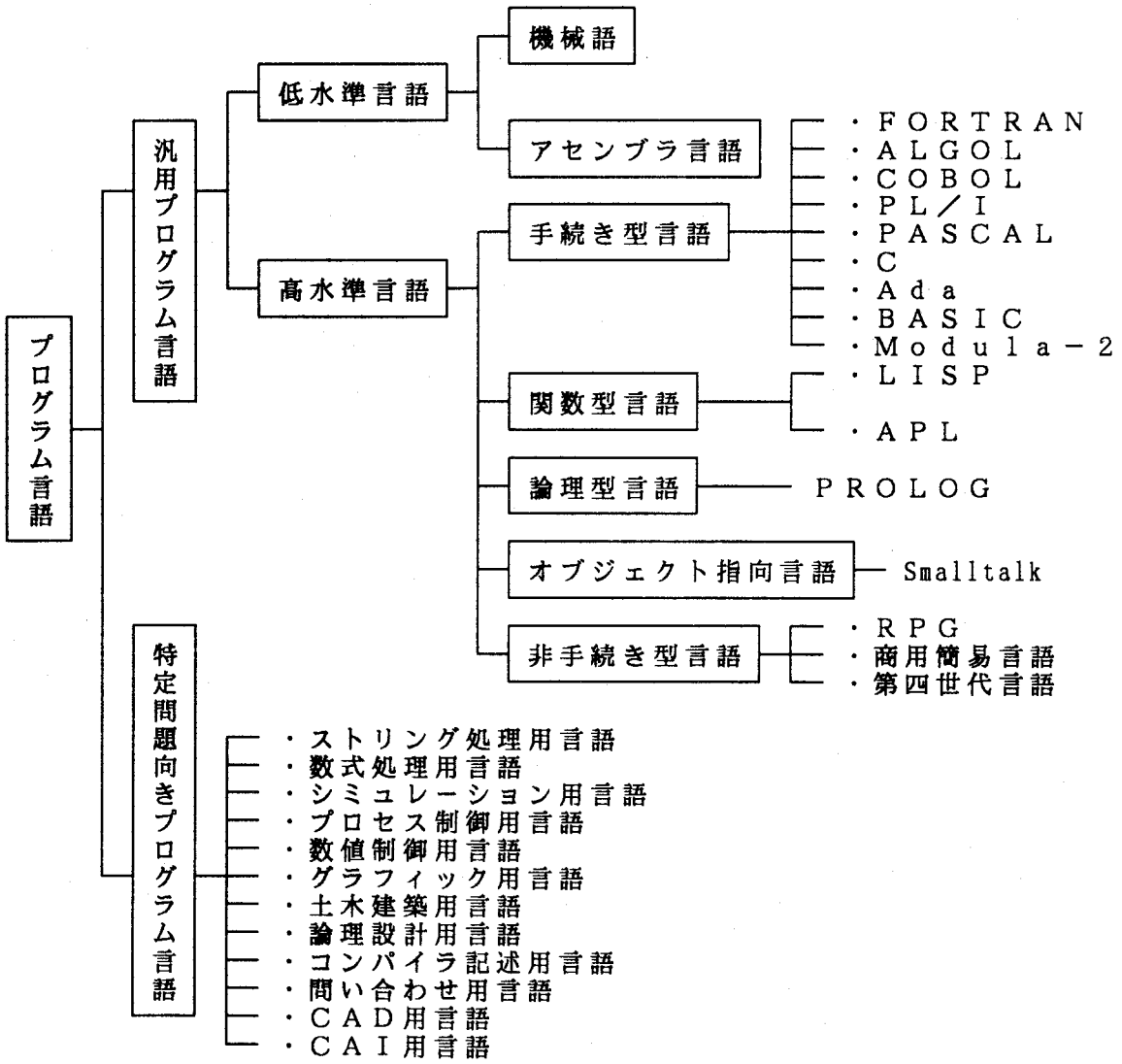
仮想記憶方式において、仮想記憶空間と主記憶空間の対応関係、および、主記憶域の使用状況の管理は制御プログラムが行っている。この仮想記憶方式としては、ページング・セグメント方式、パーティション方式、スワッピング方式などが代表なものであるが、汎用コンピュータ、小型コンピュータ、ＷＳはページング・セグメント方式が主流で、ＰＣ等ではパーティション方式まれにスワッピング方式がとられる。

ページング・セグメント方式では、仮想記憶方式の実現のために補助記憶装置が使われるが、ＰＣでは、主記憶装置と補助記憶装置との転送が極めて遅いこともあり、実記憶領域をできるだけ大きく確保するために、ＶＲＡＭ，ＲＡＭディスク、拡張メモリ等を使った仮想アドレス方式を採用している例が多い。

5．言語に関連すること

コンピュータの利用にとってプログラム言語の役割は非常に大きい。商用コンピュータが普及するにつれ、既存の言語に加え、コンピュータの利用目的、コンピュータのアーキテクチャーにふさわしい言語の研究開発がずっと行われてきた。生き残り商用化されたものと、小数のマニア（あるいは研究者）にのみ限定的に評価されたもの、成果を生かしきれず生き残れなかったもの枚挙に暇がない。

しかし、力強く生き残ってきたものは、アセンブラ言語、高水準プログラム言語、および特定問題向きプログラム言語の３種類に代表されよう。



≪汎用言語の例≫　　[出所：初級情報処理技術者育成指針－中央情報教育研究所]

言語名	主な適用分野	特徴
FORTRAN	科学技術計算	1957年に実用化され高水準言語としては最も歴史が古い。 ＩＢＭが開発し、世界的に普及してきた。 科学技術計算用プログラムの蓄積は大きい。
ALGOL	科学技術計算	1960年にヨーロッパで出現し、当初はアルゴリズムの記述を主目的とした。構造化、ブロック構造の概念を持っており、大学・研究機関で多用されていた。
COBOL	事務処理	1960年代初期に米国防省の標準的言語として開発され、現在世界的に最も普及している。データ記述と処理手続きを完全に分離したのが特徴である。
PL/I	汎　用	FORTRAN,COBOL,ALGOLの特徴を取り入れ、さらにアセンブリ言語に代わる機能をも包含し、システム・プログラム用にも利用可能とした。ＩＢＭにより開発され、現在同社の標準的言語となっている。
BASIC	汎　用	1960年代にTSS用会話型言語として米国ダートマス大学にて開発されたが1970年代後期にパーソナル・コンピュータ用に再び返り咲き、初心者向きの言語になっている。 インタプリータ型の実現方式が一般的である。
PASCAL	科学技術計算 ｼｽﾃﾑ･ﾌﾟﾛｸﾞﾗﾑ	教育用構造化プログラミングに適した言語として大学関係を中心に普及した。

言 語 名	主な適用分野	特　　徴
LISP	人工知能 ｼｽﾃﾑ･ﾌﾟﾛｸﾞﾗﾑ	関数型言語の代表格で、本来はリスト処理用である。1960年代にマサチューセッツ大学で開発された歴史ある言語であるが、近年人口知能研究の活発化にともなって多用されるようになった。
PROLOG	人工知能 ｼｽﾃﾑ･ﾌﾟﾛｸﾞﾗﾑ	フランスおよび英国で生まれ育った述語論理型言語であり、今後の知識情報処理分野での利用が期待されている。第5世代コンピュータの核言語のベースとなっている。
C	ｼｽﾃﾑ･ﾌﾟﾛｸﾞﾗﾑ	1972年UNIXｵﾍﾟﾚｰﾃｨﾝｸﾞ･ｼｽﾃﾑの下で働く言語としてベル研究所において誕生し、、ﾊﾟｰｿﾅﾙ･ｺﾝﾋﾟｭｰﾀを含むUNIXの普及によって多用されるようになった。構造化機能を持つ。
APL	科学技術計算 ｼｽﾃﾑ･ﾌﾟﾛｸﾞﾗﾑ	1950年代にすでに発想されていたと言われるが、その後機能追加やオンライン環境での利用が主としてIBMによって進められてきた。論理演算や行列演算が強力である。 数値式で扱う特殊な記号を扱える。
SNOBOL	文字処理 ｼｽﾃﾑ･ﾌﾟﾛｸﾞﾗﾑ	文字ストリングの処理を目的とした言語で1960年代初期に出現し、その後種々の機能拡張が行われテキスト処理、言語処理などに多く用いられる。
Ada	汎用 ﾘｱﾙﾀｲﾑ処理	1980年米国防省の新しい標準言語として出現したリアルタイム処理の機能を含む、大型の言語である。 従来の各種言語の特質を集大成したものとも言われ、プログラム部品の概念を持つ。

言 語 名	主な適用分野	特　　徴
Small talk	研究用 ｼｽﾃﾑ･ﾌﾟﾛｸﾞﾗﾑ	処理の対象となる一切をオブジェクトを主体としたオブジェクト指向という新しい概念に基づく対話型言語で、心理学的検討成果が加味されている。1980年代に開発された。

≪問題向きの例≫　　［出所：初級情報処理技術者育成指針－中央情報教育研究所］

ｼﾐｭﾚｰｼｮﾝ言語	ＧＰＳＳ，ＳＩＭＳＣＲＩＰＴ，ＤＹＮＡＭＯ，ＳＩＭＵＬＡ
統計処理言語	ＳＰＳＳ，ＢＭＤ
数式処理言語	ＦＯＲＭＡＣ，ＭＡＣＳＹＭＡ
数値制御言語	ＡＰＴ
構造解析言語	ＣＯＧＯ，ＳＴＲＥＳＳ，ＮＡＳＴＲＡＮ
図形処理言語	ＧＬＩＤＥ，ＧＰＬ／Ｉ
ﾌﾟﾛｾｽ制御言語	ＤＡＣＳ－ＡＵＴＲＡＮ，ＰＭ／Ｃ
ＣＡＩ言語	ＥＬＩＺＡ，ＴＵＴＯＲ

（１）低水準言語について

機械語やアセンブラ言語がこれに相当し、基本的に、１つの言語情報はＣＰＵが１つずつ順次取り出して実行する命令や、演算の対象となるデータに対応する。機械に非常に近く人間の言語から遠いところから低水準言語と呼ばれる。

①　機械語

０と１だけから情報を２進のビット列として表現することにより、命令やデータを並べるプログラムである。

プログラミングするときは、機械語コード表などを見ながら、命令の固定部分と、レジスタやアドレスなどの可変部分を人間が確認しながら２進列を順次作り出す。

機械語では、テキストの文字列としての２進数を単純に計算機の内部表現値（０、１のビット列）に変換すればよく、変換作業に時間はかからない。但し、命令やデータに誤りがないかどうかをチェックすることができないため、プログラミングミスを見つけるのに非常に時間がかかる。

② アセンブラ言語

機械語の命令毎にシンボル名を決めておくとともに、演算の対象となる主記憶域上の番地に対しても名前を付けることができるため、機械語に比べるとはるかにプログラムが作り易い。アセンブラ言語プログラムは、アセンブラがそれを翻訳（アセンブル）することにより機械語に変換される。

アセンブラではマクロを定義することにより、１つのマクロ命令を使って複数個の命令列やデータ列を作り出すことができ、プログラミングの手間を省く機能をも持ち合わせている。

（２）高水準言語

一般にコンピュータが理解しにくくなる反面、人間が理解し易くなる言語を高水準言語と呼ぶ。ハードウェアの知識がさほどなくてもプログラミングができ、また言語仕様がコンピュータに依存せずに統一的に決められるため可搬性が高い。

命令体系が事務処理計算や科学技術計算処理用に表現し易いようにできており、ファイルの扱い、アルゴリズムの記述に非常に適している。

また、構造化プログラミング（７０年代中から８０年代中にかけ注目が集められた）の流行・普及により、解読し易く、保守性の高いソフトウェアを開発することが可能である。

① 手続き型言語

処理の流れを１つずつ順に列挙してプログラムを作る。高度なプログラミングを行うためには、それなりの経験が必要である。

② 非手続き型言語

処理の実行順序を記述する必要はない。予め決められた処理方法が存在し、それに対して、何をするか、どのようなデータを用い、どのような結果が欲しいかを、パラメータの指定により入力して、目的とするプログラムを作り出す。

手続き型のプログラミングほどの専門性や経験を有していなくても業務処理ができるという利点がある。

旧くは、ＲＰＧ（Report Program Generator）があり、報告書の作成用言語として使われてきた。

近年は、表計算（スプレッドシート）、グラフ作成などを行える簡易言語、また、大型データベースを使うＣＯＢＯＬプログラミングの手間を大幅に減らすことを目的とした第四世代言語がある。

（３）特定問題向きプログラム言語

特定分野の問題に対して、汎用プログラム開発の専門家ではなく、特定業務に携わる従業者が専門的な問題を効率よくコンピュータ処理できるよう設計された言語である。

汎用プログラミング言語ほど小回りの利くプログラムを作ることはできないが、特定分野において、極めて有効なコンピュータ利用ができる点で利用価値が高い。（言語の例は既述）

①　シミュレーション用言語

モデリングを行い、コンピュータ上で模擬実験を行い系の振る舞いを解析することをシミュレーションといい、このコンピュータシミュレーションを行わせるのに用いる言語をシミュレーション用言語という。

◆離散系シミュレーション（単位時間毎の連続時間でなく、とびとびの時間間隔で値を拾う解析）として、ＧＰＳＳ，ＳＩＭＳＣＲＩＰＴ等がある。

◆連続系シミュレーション（連続する時間の経過によって動的にシステムの変化する状態を解析する）としてＣＳＭＰ，ＤＹＮＡＭＯ等がある。

②　数値制御言語

工作機に関する位置を数値情報で指令するプログラム言語である。ＡＰＴがその代表的な例である。

（４）言語の比較

一般論としての評価となるが、プログラム言語に対してはいくつかの評価の観点がある。実行性能や特殊な命令の実行を求めるもの、ある程度性能は我慢しソフトウェアの生産性・保守性等に評価を求めるもの、できるだけ日常業務の進め方に連動する少ないコンピュータ教育により業務システムが開発できる点を重視する等が考えられる。

例えば、

◆ハードウェア知識の必要性に関して

機械語＞アセンブラ言語＞高水準言語＞［高水準非手続き言語、特定問題向き言語］

◆プログラミング経験の必要性に関して

［低水準言語、高水準手続き言語］＞　［高水準非手続き言語、特定問題向き言語］

◆とっつきにくさ、むづかしいと思われがち
関数型言語、論理型言語、オブジェクト指向言語
◆事務処理システムにおける開発の生産性に関して
第四世代言語＞高水準言語＞低水準言語

≪言語のタイプによる項目別評価≫

評価項目＼言語	低水準言語	高水準手続型言語	簡易、問題向言語
処理性能	機械本来の性能	２～３割以上ﾀﾞｳﾝ	高速処理には不向
機械命令セットの多様性	全命令表現可能	多様性落ちる	多様性なし
ハードウェア知識必要性	かなり必要	高い方がよい	必要なし
プログラミング行数	極めて長い	短くなる	かなり短い
言語の記述性	低い	比較的高い	業務向けに高い
言語仕様互換性	低い	互換性高い	なし
言語習得	熟練に時間必要	ある程度時間必要	比較的容易
保守の容易性	低い	高い	高い
デバッグ効率	中程度	中程度	高い
翻訳の速度	高速	中速	低速
プログラムの可搬性	低い	高い	低い
プログラムの解読容易性	低い	かなり向上する	高い

（５）言語処理プログラムについて

言語処理プログラムは、言語変換プログラムとも呼べる。それは、ある言語仕様に基づくソースプログラムを、構文解析ルールによる字句や文の解析、および意味の解析を行い、また変換ルールに基づいて目的プログラムを作り出すからである。

但し、ソースプログラムに対して、変換された目的プログラムを作る場合と、ソースプログラムの一つ一つの文を解釈して直接マシンの動作に変えるタイプの言語もあり、以下でいくつか説明する。

① 翻訳型

(a) アセンブラ

全ての機械語命令に対してシンボルが付いており、命令の種類とそれに付随するオペランド情報を１つのステートメントとして実行順に記述する。アセンブラは、各ステートメントに関して、命令部、レジスタ部、アドレス部、オペランド部等を解決しながら、機械語に変換して目的プログラムに変換する。

大方のアセンブラは、プログラム作成の手間を省く機能として、マクロ言語機能を持っている。利用者は予めマクロを作成（一定のパターンを持つ一連の命令ステートメントの原型）登録しておき、アセンブルプログラムを作成するときに必要なマクロ名とパラメータを与えると、対応する命令ステートメントが展開される。

この機能を利用すると、大量のプログラムを開発する場合に、作成する行数が減り、またバグも少なくなる利点がある。

(b) コンパイラ

言語の種類によって仕様が統一されており、マシンが異なる毎にコンパイラが開発される。しかし、近年のコンパイラはソースプログラムの構文解析や意味解析の部分はほとんどマシンに依存しないため、マシン独立に作られている。オブジェクトコードを生成する部分のみがマシン依存となっている。

その意味で、高水準言語で記述した利用者のプログラムは互換性、可搬性、移植性が高いとともに、コンパイラ自身もオブジェクト生成部をマシン毎に書き換えれば容易にコンパイラを作ることが可能である。その代表的な言語が“Ｃ”言語である。

コンパイラにも、ＣＩＳＣ型のマシンコードを作り出すタイプと、ＲＩＳＣ型のマシンコードを作るタイプとではコンパイラの負担はかなり異なる。また、スーパーコンピュータやアレイコンピュータのような超高速コンピュータに対するオブジェクトコードの作り方も通常に比べるとかなり、ユニークである。

あまり気がつかない機能ではあるが、コンパイラはできるだけ実行効率の良い機械語プログラムを作り出すことが大きな使命の一つであり、例えば“最適化オプション”などの指定により、コンパイル時間はかかっても、命令数などの短いオブジェクトを作ることもできる。

② インタプリータ

ソースプログラムのステートメントを１つずつ取り出して文の形式や意味を解釈して、実行マシンでの動作に変えるプログラムをインタプリータという。

実行順が回ってくると毎回原始文型から機械が解釈できる形式に内部変換するため、実行速度はコンパイラの作り出す実行プログラムと比較するとかなり遅くなる。その反面、インタプリータが開始するとその場でコンパイルエラーが指摘されたり、あるいは実行途中の結果をいつでも見ることができるため、デバッグ効率が高い。

インタプリータとコンパイラ両方持っている言語でプログラムを作れば、インタプリータでデバッグを、コンパイラで作ったオブジェクトで速度の速いの実行を行うことができる。

③ ジェネレータ

パラメータを与えると、あるスケルトンに従ったプログラムを自動的に生成してくれる。生成されるプログラムは、アセンブラであったり、高級言語であったりする。作り出されたプログラムは、それ自身で完結するプログラムとは限らない。基本的には特定の処理部分だけを生成する場合が多い。

ＲＰＧは報告書作成ルーチンを生成するし、パラメータの指定により定める順序でデータの並べ変えを行うような分類作成ルーチンなどがある。

④ シミュレータ

ある機械用もしくは仮想的な機械用に作られたプログラムを、他の機械にあった命令に作り直したり、命令を直接実行するもの。

⑤ コンバータ（トランスレータ）

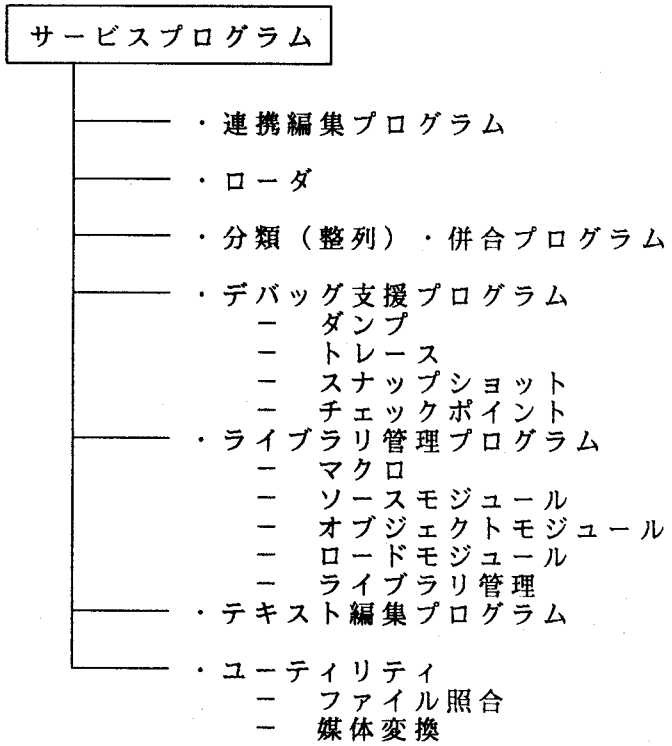
ある言語から他の言語プログラムにソースプログラムを変換するプログラムである。例としては、あるアセンブラから他のアセンブラプログラムに変換するもの、ＦＯＲＴＲＡＮからＰＬ／Ｉプログラム変換するもの、ＦＯＲＴＲＡＮからＣに変換する等が考えられる。

6．サービスプログラム

プログラム開発における労力をできるだけ軽減するためにコンピュータメーカが提供するソフトウェアで、通常ユーティリティプログラムと呼ばれる。低水準言語や、高水準手続き型言語によるプログラム作成の支援をしたり、言語プロセッサにより生成されたオブジェクトプログラムを実行可能なモジュールに編集したり、実行デバッグ支援をしたり、また、データの分類・併合、ファイルの媒体変換、プログラムライブラリの管理に役立つような、コンピュータ言語によるプログラム開発のごく基本的な部分に関する支援となるプログラムの集まりである。

つまり、オペレーティングシステムの制御プログラムや言語処理プログラムとの連携により、プログラムの実行、大量なデータの扱いなどにおいて、利用者に共通するサービスとなる機能を提供するプログラムを指している。

このプログラムは捉えようによっては、大きな範囲のプログラムに及ぶことになるであろうが、メーカが標準プロダクトとして提供したもの、計算センターなどで標準に用意した共通のプログラム資産に限定して考えることとする。



（１）連携編集プログラムについて

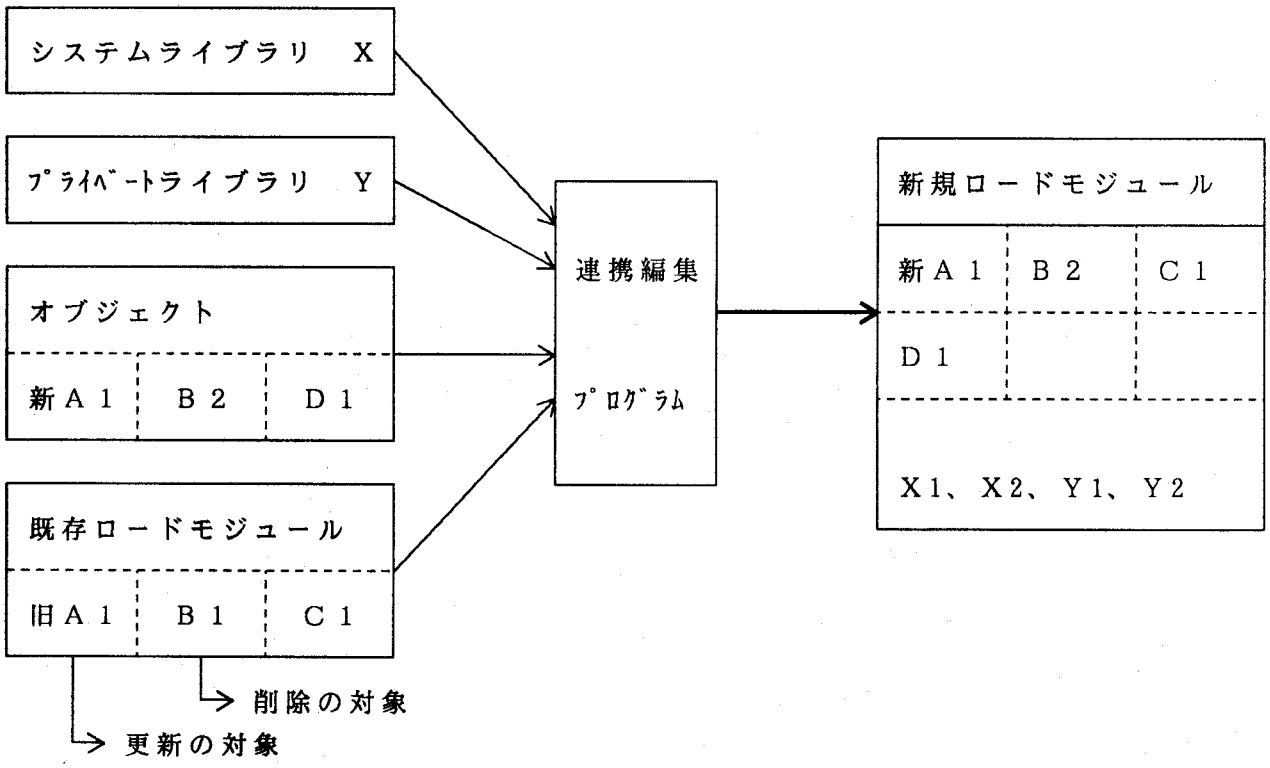
最終的には実行可能なロードモジュールを作り上げるものであるが、基本的にはオブジェクトプログラムに対する編集（結合、追加、削除、変更）を行うことが主体となる。

①　モジュールの連携

リンケージエディタに対する入力オブジェクト情報に基づき、オブジェクトプログラム同志を結合する。オブジェクトプログラムは、元の言語がＣＯＢＯＬ、ＦＯＲＴＲＡＮ、ＰＬ／Ｉ、Ｃ、アセンブラ等いずれであろうと、違いはない。元の言語がミックスされたオブジェクトの連携も可能である。

また、指定されたオブジェクトプログラムに対して結合だけをすると捉えられがちであるが、既存のロードモジュールに対するモジュールの追加、削除、変更などもできるのである。

また、各オブジェクトが使用しているライブラリをサーチし、必要なものをインクルードする。



②　未解決アドレスの決定

サブプログラムの呼び出し、外部参照変数、コモン変数、ライブラリの使用等単一のプログラム外に定義されている情報の参照に関して、そのアドレスを解決する。

---

③　オーバレー

プログラム全体が主記憶に入りきらないときに、計画型のオーバレー構造をとり、各セグメント毎にロードされる先頭アドレスを決定する。

（２）ローダについて

ローダとは、オブジェクトモジュールまたはロードモジュールを実行可能な形式に整え、主記憶域にロードするプログラムである。どのようなモジュールをロードするかにより、機能が異なる。

①　アブソリュートローダ

アドレスが全て絶対番地で解決されているモジュールについては、主記憶域の決められた番地からロードされる。

②　リロケータブルローダ

アドレスが相対番地でふられているときは、ロードされる先頭番地でバイアスされる位置にロードされる。

③　連携ローダ

オブジェクトモジュールの連携編集の機能を持ち、未解決アドレスの決定と配置領域の割当てを行いながら、実行可能なプログラムを作成すると同時に主記憶域にローディングする。

（３）分類（整列）・併合プログラム

プログラミングにおいては、通常ソート、マージと呼ぶデータ操作がしばしば行われる。ソートとは、ファイルに含まれるデータレコードをキーの大小に依って一定の順序で並べ変える操作であり、分類（整列）ともいう。

一方、マージとは、分類済の２個以上のファイルのデータレコードを統合して、キーの昇順もしくは降順に整列することを意味しており、併合ともいう。

この分類や併合を行いたい場合、

・利用者が自らプログラムを作る
・ＣＯＢＯＬプログラムでＳＯＲＴ命令をコールする
・プログラムから分類プログラムや併合プログラムを使用する

等の方法がある。

分類の方法には、いくつかのアルゴリズム（例えば、挿入法、交換法、クイックソート等）があり、どのような場合にどのようなアルゴリズムを用いるべきか、実際にプログラミングを行って確認する必要がある。

実際にデータレコードの分類を行う場合、データレコードの量によってどういう計算法を用いるかを考える必要がある。

データレコード量が少ないときは、データの並べ変えを主記憶域内で行うのが一般的で、これを**内部分類**（Ｉnternal ｓort）という。
また、大量のデータレコードを分類する場合、分類のために、磁気テープやディスク装置を用いる必要があり、これを**外部分類**（Ｅxternal ｓort）という。

（４）デバッグ支援プログラム

プログラムに存在する誤りを探し、それを取り除くことをデバッグという。デバッグを行う方法として、

・プログラムソースリスト、プログラムソース編集画面を目視により誤りを発見する
・プログラムの実行結果を頼りに誤りの所在を推定する
・デバッグ支援プログラムの助けを借りて複雑な根の深い誤りを発見する

等が考えられる。

ここでは、コンピュータによるデバッグ支援について若干触れる。このデバッグ支援を実現する仕掛には２通りの方法がある。

① プログラムの中にデバッグ機能が埋め込まれているケース

(a) プログラムのステートメント中にデバッグｗｒｉｔｅ文や途中情報のｒｅａｄ文がプログラミング時に組み込まれている。これは、デバッグの仕掛をプログラム作成者もしくはデバッグ担当者がソースプログラム中に組み込んだものである。プログラムに誤りがなくなるにつれ、デバッグに関連する情報が煩わしくなるので、あまり多用することを奨めない。

(b) プログラムのコンパイル時に“デバッグ”オプションを付けると、コンパイラは目的モジュールの中にデバッグ支援プログラムが参考にするような情報を作り出す。プログラムは大きくなるが、プログラムの実行が始まると自動的にデバッグに参考となるような各種情報が出力される。

② プログラムの外でデバッグ対象プログラムを制御するケース

①の(b) と同じように、プログラムのコンパイル時に“デバッグ”オプションを付ける。実行にあっては、どのようなデバッグを行って欲しいかを予めファイル化しておくか、デバッグ支援プログラムが動作を開始したら、その時点でデバッグに必要な情報を与えるか

のいずれかの方法をとる。

デバッグの手段として、以下の機能がある。

◆ダンプ機能

－　メモリ（コア）ダンプ　　指定された記憶領域の内容を指定サイズ分出力する

－　スナップショットダンプ　特定の場所にきたり、特定の命令が実行されたりしたときに、関連する内容（レジスタ、変数、主記憶領域等）を出力する

◆検査機能　　プログラム中にチェックポイントを設け、プログラムの動作が正常か否かを確認する。チェックポイントに到達したとき、必要な情報（通過回数、主記憶域等）を出力する。

◆追跡機能　　トレースともいい、指定された命令が実行されるたびに、所在、命令、レジスタ、変数の内容などを出力する。

（５）ライブラリ管理プログラム

ライブラリにプログラムを登録、交換、追加、削除を行い、ライブラリの維持・管理を行う。さらに、ライブラリ間でのプログラムのコピー、ライブラリ中のプログラムの一覧表のリストアップ等を行う。

登録の対象となり得るプログラムとしては、

◆マクロライブラリ　－　アセンブラ言語、高水準言語等で使用するマクロを管理

◆ソースモジュールライブラリ　－　ソースコードのままプログラムを管理

◆オブジェクトモジュールライブラリ　－　オブジェクト形式のプログラムを管理

◆ロードモジュールライブラリ　－　ロードモジュール形式のプログラムを管理

ライブラリの管理は、区分編成のデータセット管理を行うのが一般的で、登録したプログラム情報はディレクトリ領域に、個々のプログラムはメンバとして格納される。

（６）テキスト編集プログラム

プログラム、データ等、テキスト形式（文字列）のファイルに対して、内容の変更、削除、追加を行うプログラムで、ほとんどの場合、対話型で編集動作する。

テキストのどのような単位で編集を行うかは、編集をする端末の種類などにより異なり、レコードエディタ、フルスクリーンエディタ等がある。

また、文字や行を編集の単位とするプログラムの他に、プログラムの構造に対応した編集機能を持つ構造型エディタが開発されている。

7．ユーティリティプログラム

基本ソフトウェアの内、オペレーティングシステムを除いた部分を全て総称してユーティリティプログラムと呼ぶことも可能であるが、さらにプログラムの開発やテストあるいは登録・維持・管理に直接関連する部分を除く、主としてデータセットに関する内容照合や、媒体の変換などを司るプログラムをユーティリティと呼ぶこととする。

プログラムユーティリティには、通常のコンピュータ利用者がデータセットに関して、内容の照合や媒体の変換あるいは複写等を行うためのプログラムと、オペレーティングシステムをメインテナンスをするシステムプログラマ等によってシステム管理回りの仕事に利用するプログラムとがある。

（１）データセットユーティリティ

プログラマやオペレータが使用する。（プログラミングにはよらず、ＪＣＬを使用して実行できる）

・内容の確認
・コピー
・データセットの媒体変換（例：ディスク→磁気テープ、磁気テープ→カード、ディスク→ディスク）
・データセットのアロケーション
・データセットの削除

（２）システムユーティリティ

システムプログラマがシステム回りの生成、維持、管理に使用する。

・ＤＡＳＤのボリュームの初期設定
・ＤＡＳＤボリューム内容の印刷

指導上の留意事項

本章は２種試験における広い出題の範囲に関わりを持っている。メインフレームの基本ソフトウェアを中心として指導するため、その使用環境が望まれる。習うより慣れる部分が多く、できるだけマシン時間をとるように、また効果的にマシン実習を行うことが肝要である。

メインフレームに触れる環境がないときは、本章はほぼ知識の伝達になりなりかねない。たとえＷＳやＰＣがあっても、オペレーティングシステムの考え方や言葉使いがかなり異なるため、指導に混乱を来すこともありうる。

そのような場合には、この部分は、２種試験の出題範囲にからむからということで、机上での演習（過去の出題例を中心に）と、ディスカッションを主体に展開すべきであろう。

勿論、ＷＳやＰＣでもオペレーティングシステムの理論を説明することはできるから、メインフレームと共通する部分については、コンピュータに触れながら原理を考えさせる工夫が求められる。

用　語

オペレーティングシステム、制御プログラム、基本ソフトウェア、カーネル、
デバイスドライバ、割込み処理ルーチン、ＲＡＳＩＳ、監視プログラム、
ジョブ管理、タスク管理、データ管理、記憶域管理、通信管理、保全管理、運用管理、
多重プログラミング、バッチ処理、タイムシェアリング処理（ＴＳＳ処理）、
オンラインリアルタイム処理、ジョブ制御、ジョブ制御言語（ＪＣＬ）、
マスタースケジューラ、ジョブスケジューラ、リーダ、イニシエータ、ターミネータ、
ライタ、ディスパッチャー、タスクスケジューリング、割込み制御、内部割込み、
機械割込み、プログラム割込み、監視プログラムコール、外部割込み、
低水準言語、高水準言語、手続き型言語、非手続き型言語、特定問題向き言語、
サービスプログラム、連携編集プログラム、ローダ、分類（整列）・併合プログラム、
デバッグ支援プログラム、ライブラリ管理プログラム、テキスト編集プログラム、
ユーティリティ、ダンプ、チェックポイント、トレース

第2種情報処理技術者試験

本章で、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、“ソフトウェアの基礎知識”に関する１２項目のうち５項目が関係しており、重点的に学習する意義は深い。

①オペレーティングシステム（制御プログラム）に関すること。
　オペレーティングシステムの目的、機能、役割、歴史など。

②プログラム言語に関すること。
　アセンブラ、コンパイラ言語、インタプリータ、ジェネレータなどの言語の特徴、歴史、用語など。

④ユーティリティに関すること。
　各種ユーティリティプログラムの使用目的、使用方法、用語など。

⑧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など。

⑨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７章　通信ネットワークに関連する事項

指導目標

情報処理環境は、コンピュータと通信ネットワークとの複合的な集合体となっており、各所にソフトウェアが介在するとともに、それらを利用する人間が存在する。通信ネットワークは、コンピュータとコンピュータ、あるいは人間とコンピュータを繋ぐことにより優れたシステム化技術を提供する基礎となるものである。

情報処理技能者養成施設では、大規模な情報処理システムを利用できる環境にはないが、一般にはコンピュータと通信技術の統合により、高度なサービスを提供することが可能となることを強く認識させる必要がある。

本章では、通信技術により可能となるコンピュータの利用形態、システムの構成、通信ネットワークシステムについて学習する。

○情報処理サービスの方式・・・・・・・・通信ネットワーク技術によって、どのようなコンピュータ処理が可能となったか、およびサービスイメージとソフトウェアの介在状況について理解する

○コンピュータシステム構成・・・・・・・計算機システムの信頼性、経済性、運用の容易性をどう実現するか、分散処理形態の効果（ねらい）について理解する

○通信ネットワークシステムとサービス・・通信ネットワークの役割と利用例、データ伝送を確実に行うための情報表現、送信・受信側の手順、各種のコンピュータ同志、コンピュータと端末とが相互交信するためにそれぞれの間で必要な標準的な約束ごと、ルール、等について理解する

通信については、一方的な解説になりがちであり、できれば養成施設内電子メール、パソコン通信等を利用して通信の利用価値などを実感することが望ましい。

内容のあらまし

内　容	説　明	議　論	机上実習	計算機実習
情報処理サービスの方式	・遠隔地からのコンピュータ利用 ・ＬＡＮによる情報交換利用 ・物理的なネットワーク環境	・ｽﾙｰﾌﾟｯﾄ ・ﾀｰﾝｱﾗｳﾝﾄﾞ・ﾀｲﾑ ・ﾚｽﾎﾟﾝｽﾀｲﾑ 等について議論	↑	ＰＣ，ＷＳ汎用機が有ればＴＳＳ、ﾘﾓｰﾄﾊﾞｯﾁ対話処理が可能
コンピュータのシステム構成	・ｼﾝﾌﾟﾚｸｽ構成 ・ﾃﾞｭﾌﾟﾚｸｽ構成 ・ﾃﾞｭｱﾙ構成 ・ﾏﾙﾁﾌﾟﾚｸｽ構成 ・分散処理ｼｽﾃﾑ	計算機ｼｽﾃﾑの信頼性、分散処理のねらい効果について議論	第２種情報	特になし
通信ネットワークの役割	ｺﾝﾋﾟｭｰﾀ環境と通信機能の融合により可能となるシステム例、通信ソフトの役割について説明	情報処理と通信機能の融合が、どのようなｿﾌﾄｳｪｱ的な仕掛により可能かを議論	処理技術者	特になし
データ伝送	データ伝送に関する機能、実現方式、通信方式、誤り制御等について説明	特になし	試験等	ＴＳＳ環境
伝送制御手順	伝送制御手順、通信プロトコルなどについて説明	伝送制御手順の確立により、どんなｺﾝﾋﾟｭｰﾀ処理が可能となったか	↓	ＴＳＳ環境

内　　容	説　明	議　論	机上実習	計算機実習
通信ネットワーク技術の利用	ﾈｯﾄﾜｰｸの形態、ﾈｯﾄﾜｰｸｼｽﾃﾑの構成要素、ＬＡＮﾈｯﾄﾜｰｸ体系について説明	ネットワークの出現によりｺﾝﾋﾟｭｰﾀ環境はどう高度化したか議論	特になし	ＬＡＮ ＰＣ，ＷＳ ﾌﾟﾘﾝﾀ,FAX, ﾜｰﾌﾟﾛ,ﾓﾃﾞﾑ

指導内容

1．情報処理サービスの方式

コンピュータのサービス方式は、中央演算装置（ＣＰＵ）の高速化、実装メモリ量の増加、ディスク容量の大型化、通信回線速度の高速化や高信頼化によりサービス形態に大きな変化をもたらされた。また、高性能ディスプレー、高解像度グラフィック装置の登場により、タイムシェアリングサービス機能も充実し始めた。

一方、オフコンやミニコンの、よりユーザ業務へのサービスの高度化により大規模な情報処理部門の管轄下でなくとも部門独自に計算処理を行うことが可能となった。

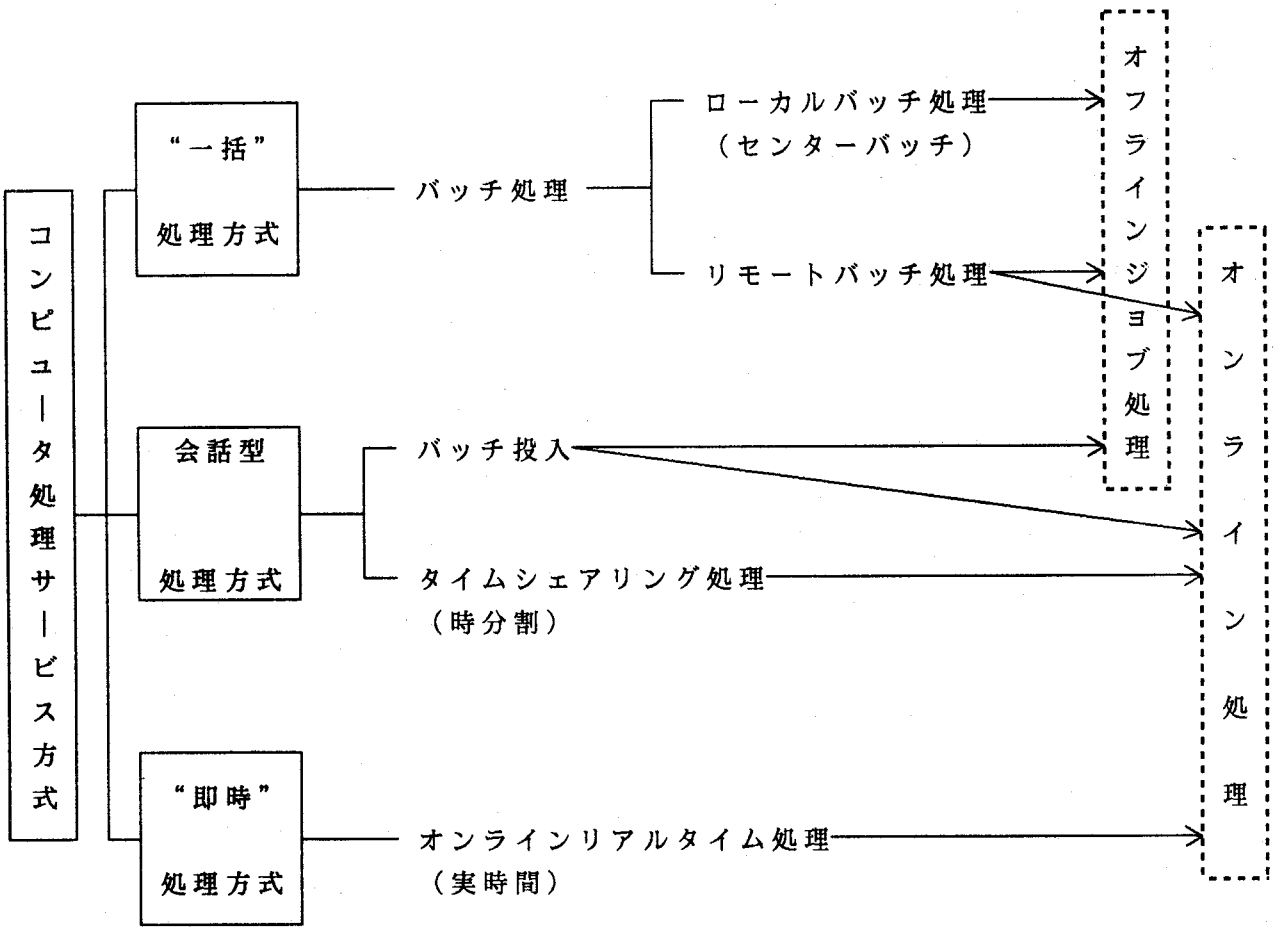
さらに会話処理を中心として進めるマンマシンインタフェースに優れたＰＣ、ＷＳの登場により、オフィスにおける事務処理、研究所におけるラボラトリーオートメーション、エンジニアリング業務の知的高度化と効率化に大きな貢献をしている。また、それらとローカルエリアネットワークや公衆回線との結合により、パソコン通信、社内メール交換、情報資産の協同利用が促進され、時間と空間の距離があるサイト間で情報の交換が可能となっている。

（１）汎用コンピュータおよび周辺機器を中心とした情報サービス

汎用機は、企業における勘定系の基幹業務、公共団体における基幹サービス業務、大学などにおける大型計算業務、情報サービス会社や機関における情報のリアルタイム提供サービスに利用されてきている。

汎用コンピュータで処理される業務は種々存在し、また、処理要求はそれを必要とする人間の存在する場所から出され、処理結果をそこへ返送することが好ましい。このように汎用機においては、処理の種類や発生地点に広く対応可能とするために、また、高価なマシンであるために、極力それを有効利用すべく種々のサービス方式が提供されている。

以下で、まず、コンピュータ処理サービス方式について図にまとめ、さらにそれぞれについて簡単な解説を行う。



（注）オフライン処理とオンライン処理の厳密な区別は以下の通りである。

(a) オフライン処理 － 基本的には、コンピュータセンターのジョブ入出力装置に対して処理対象のジョブが投入され、バッチジョブとして処理された後、結果がセンターの印刷装置などに一括出力され、センター窓口より利用者に返送される。

(b) オンライン処理 － 通信回線などを経由してセンターマシンにジョブが投入される。処理された結果はやはり通信回線を経由して利用者の端末などに直接返送されるため、この方式をオンライン処理と呼んでいる。
センターマシンに入ったジョブは、バッチ処理されるもの、時分割により処理されるもの、即時処理されるものとがあり、この辺の厳密な違いをはっきり説明する必要がある。

① ローカル処理、リモート処理

利用者が、コンピュータの存在する場所にいる場合（オンサイト）と、そこから離れた場所にいる場合（オフサイト）とで処理の対象となるプログラムなどの入力場所や計算結果の出力場所が異なる。コンピュータに直結した入力装置や出力装置を介して計算処理を行う場合をローカル処理とよび、通信回線等を介して計算処理を行う場合をリモート処理という。

② 一括処理（バッチ処理）、即時処理（オンライン・リアルタイム処理）

バッチ処理（一括処理ともいう）とは、処理の対象となるプログラムやデータを一定の量あるいは一定期間中央の汎用コンピュータ（ホストコンピュータともいう）に貯めておき、処理対象がまとまった段階でそれらの一連の処理を一括して実行する形態をいう。

バッチ計算処理の入力は一般にジョブと呼ばれる単位で行われ、コンピュータに直結した入力装置からエントリーする場合と、遠隔地から通信回線を利用してエントリーされる場合（これをリモートジョブエントリー［ＲＪＥ］と呼ぶ）とがある。

一括処理の対象となるプログラムやデータは、汎用コンピュータの周辺機器（カードリーダなど）から入力される場合（ローカル入力）と、通信回線を通して入力される場合（リモート入力）とがある。このような処理対象の例としては、大型の計算を伴う給与計算処理、大量データの統計解析処理、科学技術関連のデータ解析処理などが考えられ、高速の計算能力を活用することが大きな目的である。

一方、即時処理（オンラインリアルタイム処理、実時間処理などともいう）とは、データの入力やトランザクションの発生時点で即必要な処理を行って結果を返答する方式である。この場合、トランザクションは通信回線などを介して外部から発生し、元々その処理に求められている時間以内に計算が完了されなければならない。

このような処理対象の例としては、列車、飛行機の予約業務、ホテル、旅館の予約、金融機関の現金引き出し、預貯金等のオンラインシステムが典型的な例である。この処理システムの特徴は、処理要求を発してからできるだけ即時に処理が行われ、短時間に応答がなされる必要がある。要求を出すサイトは極めて多数であり、また応答内容には高度な信頼性も要求されるため、大型のコンピュータが中央に設置され、各地に通信回線を通じて多数の端末が接続される。

③ 対話処理（タイムシェアリング処理）

コンピュータにおけるタイムシェアリングとは、ある計算処理において途中でコンピュータが周辺装置などと入出力を行う場合、中央演算機構に空きができるため、その計算処理を中断して他の計算処理を開始することにより、高価な計算機の演算機構の稼働率を上げようとする仕掛けであり、通常これを時分割とよんでいる。

この原理を応用して、人間と計算機との対話によって計算が進められる対話処理環境（通常ＴＳＳと呼ぶ）を実現している。これは対話処理用端末から計算機を一人の利用者が独占でき、それが同時に複数台の端末で可能であり、プログラムの開発環境、技術者用の比較的小さな数値、統計計算処理に適した利用形態である。

また、ＣＡＤやグラフィック端末をＴＳＳ端末として利用することにより、設計、シミュレーション業務の効率化にも非常に役に立っている。

④　オフライン処理、オンライン処理

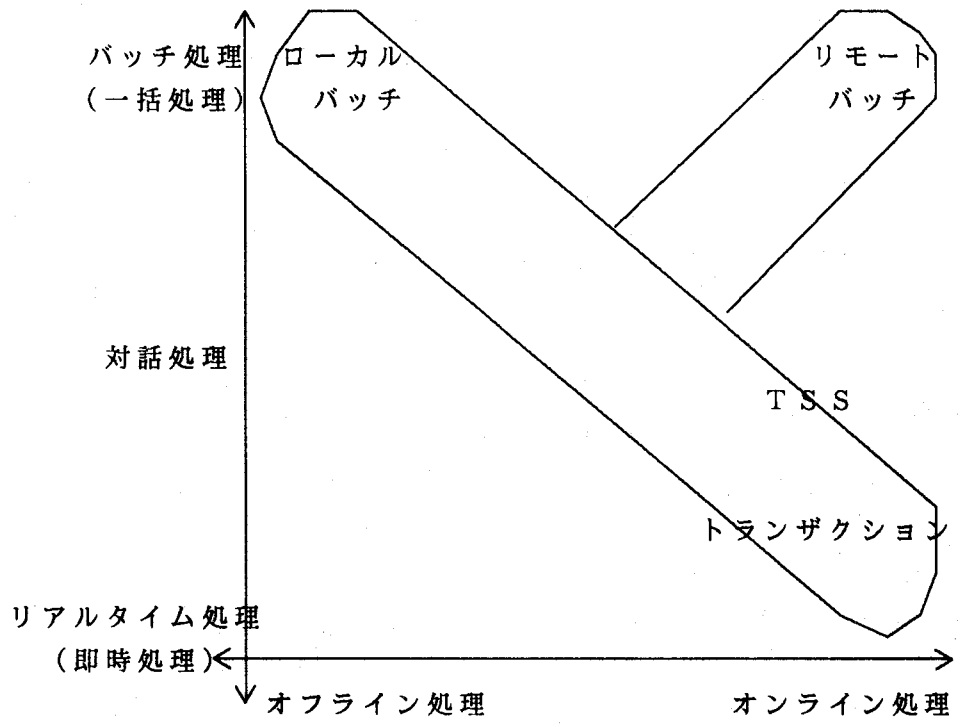
コンピュータが受け付けた計算処理は、オフラインもしくはオンラインいずれかでの処理が行われる。どちらの処理が行われるかは、アプリケーションの種類や、人手が介在するかどうかにより区分けされる。

オフライン処理は、計算処理の受付はコンピュータに直結する入力装置より行われ、計算処理結果はやはり周辺装置としての磁気ディスク、磁気テープ、さらにはプリンタなどに出力される。この過程では人手を介すことが特徴である。

オンライン処理では、計算処理の入出力過程は人手を介さずに行われる。即ち、計算処理要求の発生場所と結果の返送が遠隔地にある場合にとられる形態である。

オンライン処理とリモート処理は通信回線を利用してデータの送受信が行われることは共通しているが、中央処理装置における処理方法は対照的である。前者をオンライン・リアルタイム処理、後者をオンライン・バッチ処理（リモートジョブエントリー）という。

ここで汎用機における情報処理サービスを形態別に分類図示すると下記のようになる。



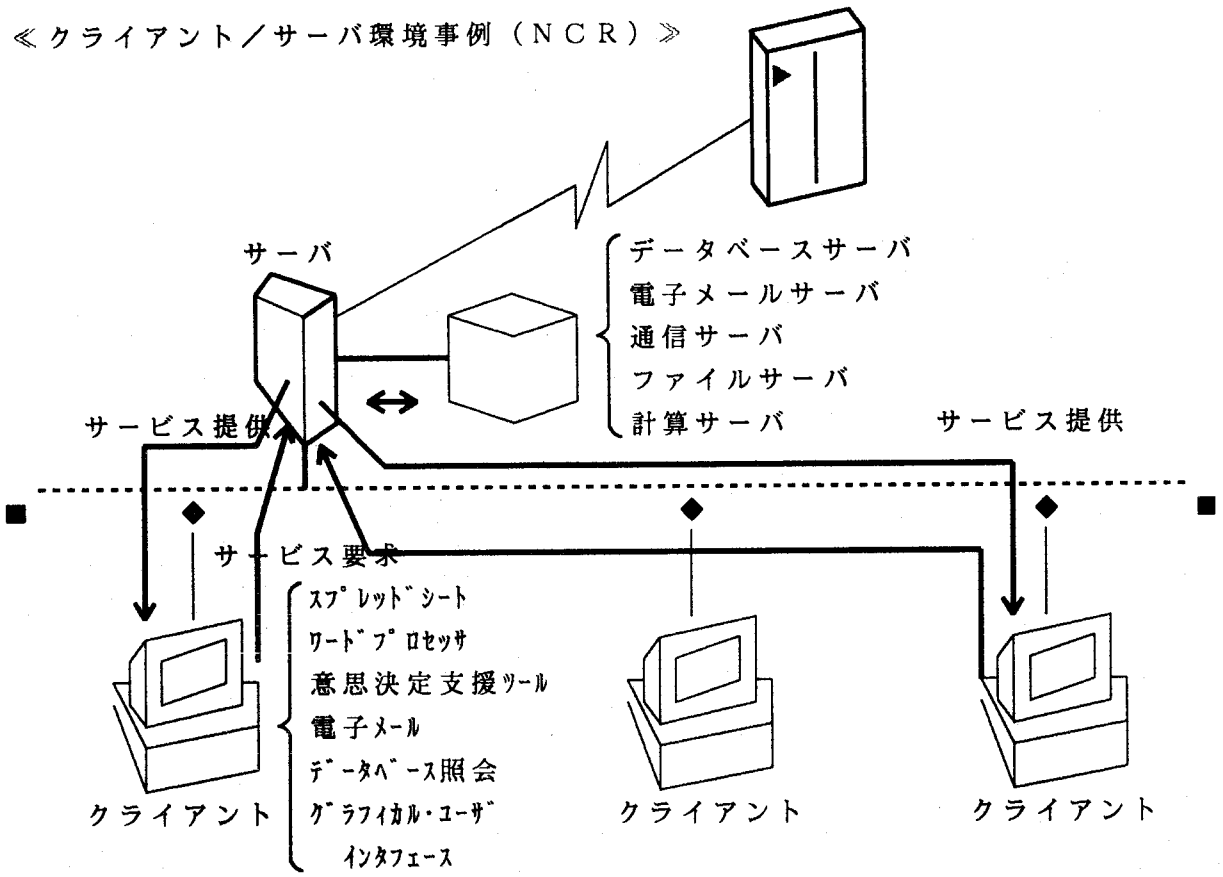
（２）ローカルエリアネットワーク・コンピュータ環境

システム	内　　容	システム資源
ソフトウェア開発環境	・システムソフトウェア開発 ・エンジニアリング用ソフト開発 ・マイコン用ソフトウェア開発	ＬＡＮ、ＰＣ、ＷＳ、ＤＢＭＳ ＣＡＳＥツール、各種ソフトウェア開発支援ツール
研究開発環境	・モデリング ・シミュレーションと解析、評価 ・実験データ取得、内容解析	ＬＡＮ、ＰＣ、ＷＳ、画像処理システム、シミュレーションパッケージ
ＣＡＤ／ＣＡＭ／ＣＡＥ	・デザイン ・ＮＣコントロール	ＬＡＮ、ＰＣ、ＷＳ、高性能グラフィック端末、２・３次元ＣＡＤソフト、ＣＡＭｱﾌﾟﾘｹｰｼｮﾝ
オフィスオートメーション	・オフィスの一般事務 文書作成、事務連絡、伝票発行 予算管理、各種集計	ＬＡＮ、ＰＣ、ワープロ、表計算ソフト、ＤＢソフト、電子メール

（３）クライアント／サーバ・コンピュータ環境

メインフレーム中心のコンピュータ環境から、ダウンサイジングの浸透により、コンピュータ形態が次第にエンドユーザ主体のシステムに変わりつつある。クライアント／サーバ・コンピュータ環境は９０年代を代表するコンピュータ処理形態として期待の大きく、またエンドユーザのコンピューティングニーズを最も満足する形で実現できるものとして普及が進んでいる。

≪クライアント／サーバ環境事例（ＮＣＲ）≫

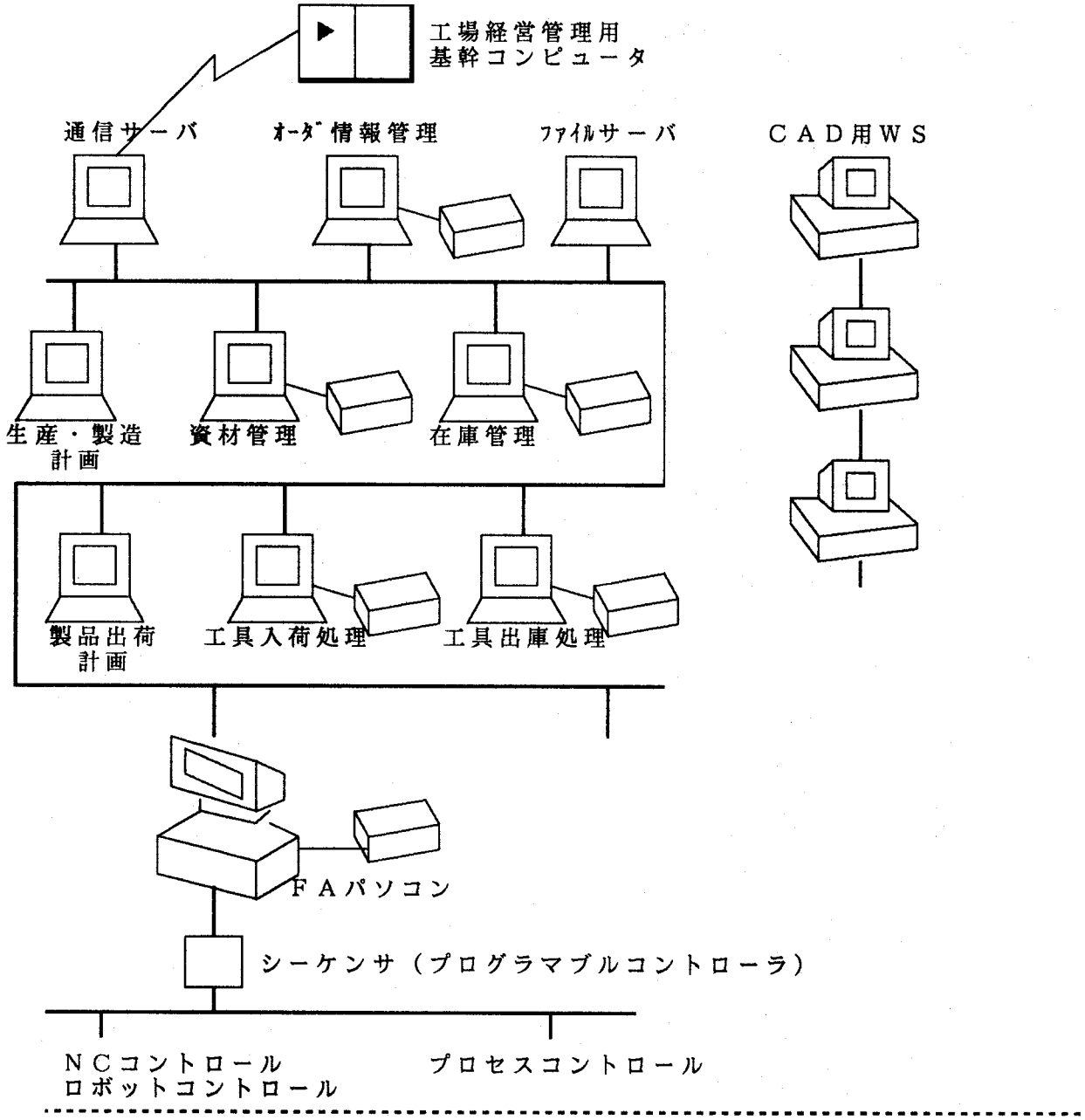


上記構成を一般に、業務部門コンピュータ環境と呼び、業務部門専用のサーバマシンとクライアントマシンとからなる。業務部門の利用者は、部門専用のデータベースのアクセス、通信サーバを介してのホストコンピュータのデータベースのアクセスを、またクライアントマシン自身では個人データベースなどを利用することができる。

このシステムでは、１つのアプリケーションをクライアント（ＰＣやＷＳ等）上で実行するフロントエンド部（サービスの要求側）とサーバ上で実行するバックエンド（サービスの提供側）に分割し、必要に応じ両者が連携をとることで、ネットワークで結合された２つのコンピュータを１つの仮想的なコンピュータのように操作できる。

（４）ファクトリーオートメーション

工場における各種の自動化が進み、さらに情報処理技術の革新によりさらに高度な生産性の向上、技術の高度化が図られている。特に、ＣＡＤ、ＣＡＭ技術、コンピュータグラフィックス技術、工場フロアレベルの通信ネットワーク技術、マイクロコンピュータ用ソフトウェア開発の支援技術などの発展に依るところが多い。



2．コンピュータのシステム構成

コンピュータのシステム構成は、そのシステムの利用者が、

○　どのくらいの量のデータ処理を行うか（アプリケーションの規模や多重性）
○　どのくらいの処理速度（高速性、性能など）を求めるか
○　どのくらいのターンアラウンドタイムでの処理結果の提供を望むか
○　どのくらいの応答時間で問い合わせなどに応えて欲しいか
○　どのくらいの信頼性を求めているか
○　障害の発生に際して、どのような回復（早さ、正確さ）を必要とするか

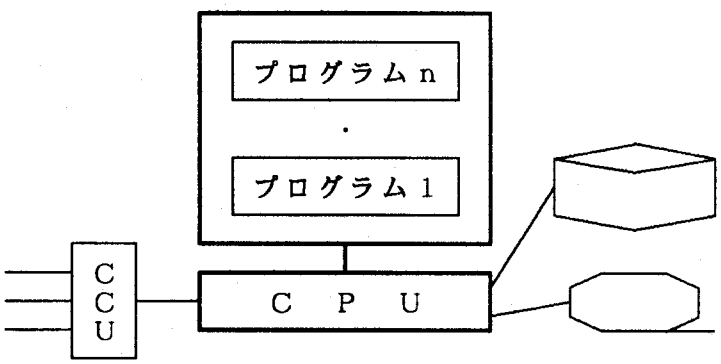
等により決定されなければならない。以下で、主として信頼性を高めるための代表的なシステム構成について概観する。

システムの構成	特　徴	短　所
シンプレックスシステム ＣＰＵ、主記憶装置、入出力装置、通信制御装置に関し予備を持たず必要最小限の機器構成をとる	バッチ処理、リアルタイム処理が共存できる。安価に調達できる構成	機器構成に冗長性がなく、どの機器が故障してもシステム停止を免れない
デュプレックスシステム ＣＰＵをはじめ各装置が予備をもつ構成。一方に故障が起きると他方に切り換え、処理を続行する。	通常は主系がﾘｱﾙﾀｲﾑ処理を、従系がﾊﾞｯﾁ処理を行い、主系故障時に従系に切り換える	利用率の低い系があればコスト高にはなる
デュアルシステム ＣＰＵ、その他の装置を二重に持ち２系列で同一処理をし、結果照合を行う。故障時は１系列で処理する	両系での結果照合によりﾃﾞｰﾀ処理の正確性が増し、信頼性が高まる。	二重の装置を同目的に使うため、コスト高となる。
マルチプロセッシングシステム ＣＰＵを複数個持ち、主記憶域、ファイルを共有する。処理効率が高く、１台のＣＰＵだけでも処理可能	１つのＯＳで複数個のＣＰＵにより処理する。多量処理に向き、経済的で信頼性も高い	処理量が少なくなるとＣＰＵその他がアイドルとなりコスト高になる
タンデムシステム 通信処理制御やﾃﾞｰﾀﾍﾞｰｽ、ﾌｧｲﾙ等の処理をCPUから切り離し、負荷分散によりﾃﾞｰﾀ処理能力を高める構成	通信処理をﾌﾛﾝﾄｴﾝﾄﾞﾌﾟﾛｾｯｻで、ﾃﾞｰﾀ検索処理をﾊﾞｯｸｴﾝﾄﾞﾌﾟﾛｾｯｻで代行。ＣＰＵ負荷が軽減	

（１）シンプレックスシステム

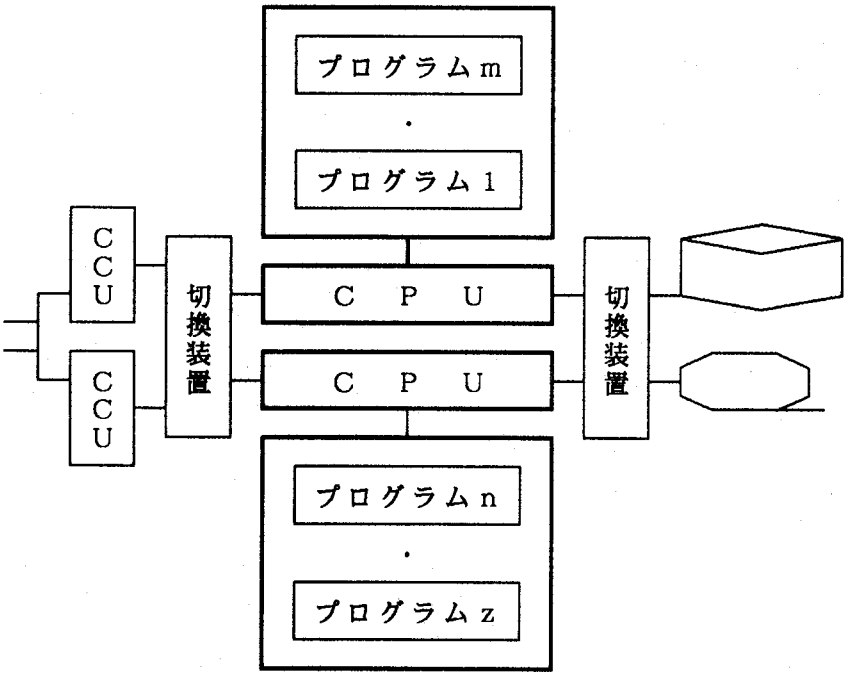
どの機器にも冗長性を持たせない単一型のシステム構成である。

ＣＣＵ：回線制御装置
Communication Control Unit

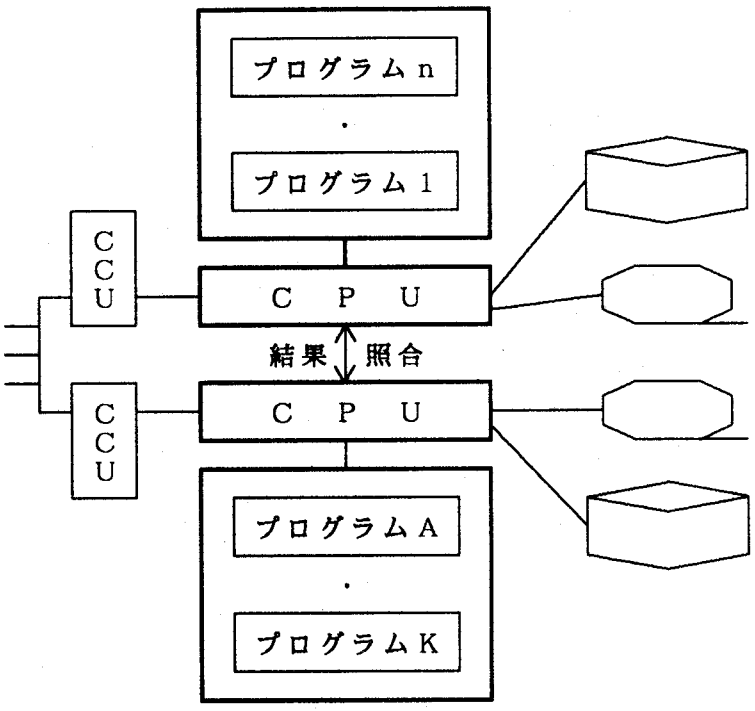


（２）デュプレックスシステム

オンライン処理などに重要と思われる機器について、二重に持つ。重要かつ優先度の高い処理を行う方を主系とし、通常は主系でオンラインリアルタイム処理などを行う。他の従系では優先度の低いバッチ処理を行う。主系が故障した場合、従系のバッチ処理を切り離し、そこでリアルタイム処理を続行する。その間主系の方は修理を行う。

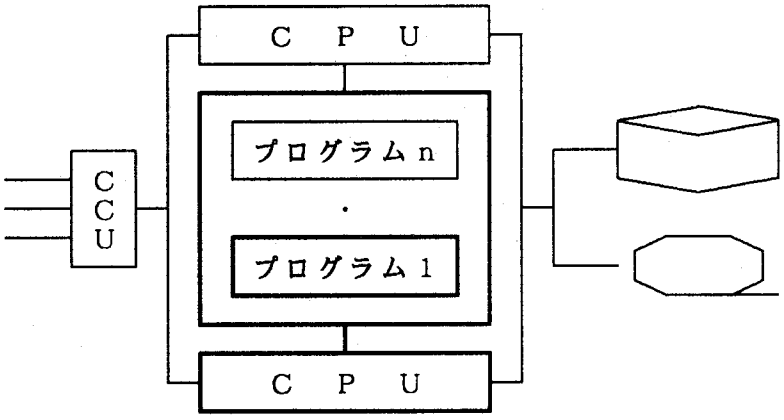


（３）デュアルシステム



（４）マルチプロセッシングシステム

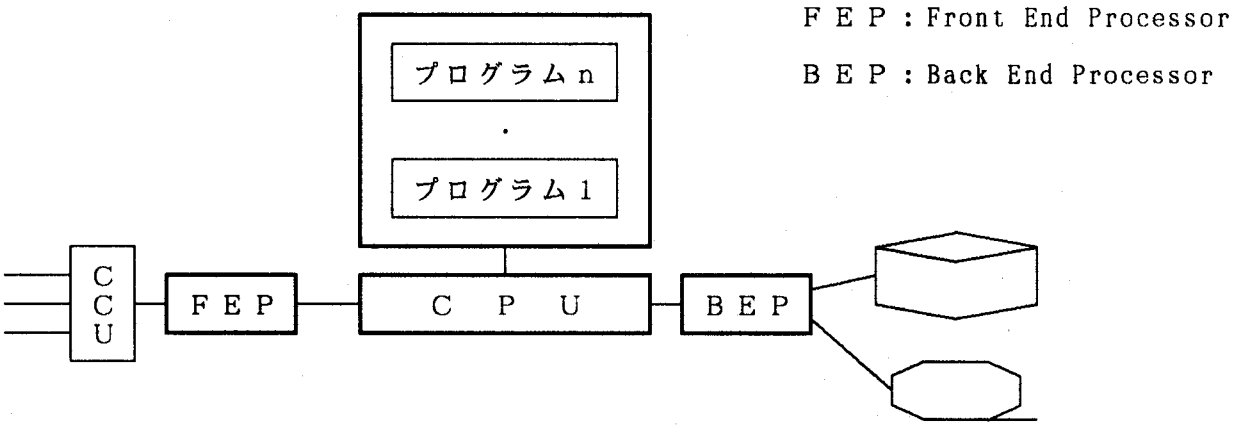
ＣＰＵを複数個持ち、１つのオペレーティングシステムの制御のもとにそれぞれのＣＰＵに対して処理をすべきプログラムが割り当てられ、効率の高い計算処理が行われる。各ＣＰＵは主記憶装置、入出力装置、通信制御装置などを共有する。どれか一つのＣＰＵが正常であれば、効率は別として処理は続行できる。



（５）タンデムシステム

データ伝送制御、データの送受信をＣＰＵ側で行わず、一つの独立した計算機能を持つプロセッサ（フロントエンドプロセッサと呼ぶ）に代行させる。

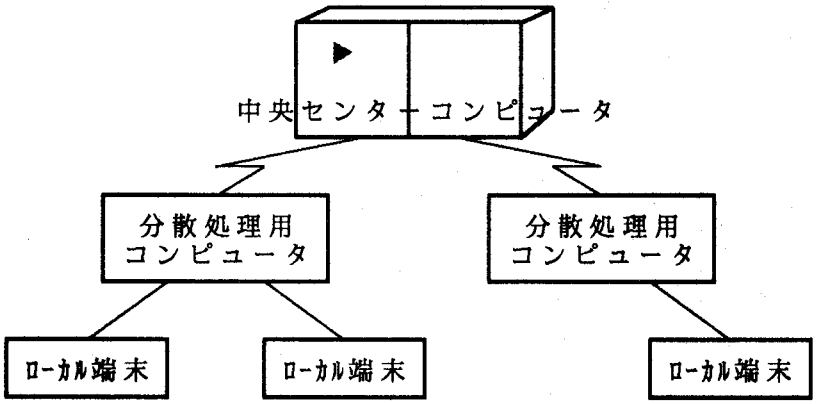
ファイルの入出力、データの検索等をＣＰＵ側で行わず、一つの独立した計算機（バックエンドプロセッサと呼ぶ）に代行させる。



以上のようなシステム構成の他、中央コンピュータの負荷を軽減させたり、手元で独自の計算処理を行うことを目的とした「分散処理システム」構成を組むこともできる。

中央コンピュータによる集中処理に比べ、負荷分散の他に、いくつかの利点がある。

・中央コンピュータに障害が発生しても、ローカルデータの処理は続行できる
・通信回線を経由した多量のデータ伝送にかかるコストを削減できる
・業務部門特有の計算処理を効率よく行える



3．通信ネットワークの役割

情報通信環境では、コンピュータとネットワーク回線がシステムの基本要素であり、ネットワークの中核に位置づけられる部分にデータベースが構築され、それをアクセスして業務システムに応用するアプリケーションが従来から開発されている。この形態もホストコンピュータと各遠隔地にある端末とによる集中処理型のコンピュータ環境では、複雑なものではなかった。このような場合、メーカの提供するホストコンピュータ用のデータベース・ソフトウェアとデータ通信ソフトウェアを利用してアプリケーションを作ればよかった。

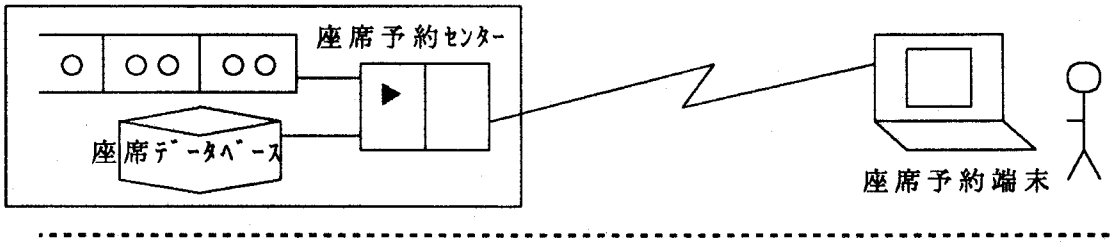
しかしながらコンピュータが多様化し、ネットワーク環境では一社のコンピュータ機器ですまされる時代が終焉して、種々のメーカのコンピュータやＯＡ機器間での情報の交換、データの相互利用、コンピュータ資産の協同利用がなされなくてはならない時代に入っている。

コンピュータの有効利用は、単なる熟練職人的なプログラマーあるいは芸人的プログラミング技法によって可能であった時代から、いかにコンピュータ資源を多数の人が協同利用し、また技術者間のネットワークを利用した情報交換を促進することにより、より高度で知的な仕事を行いうるかに変わってきている。

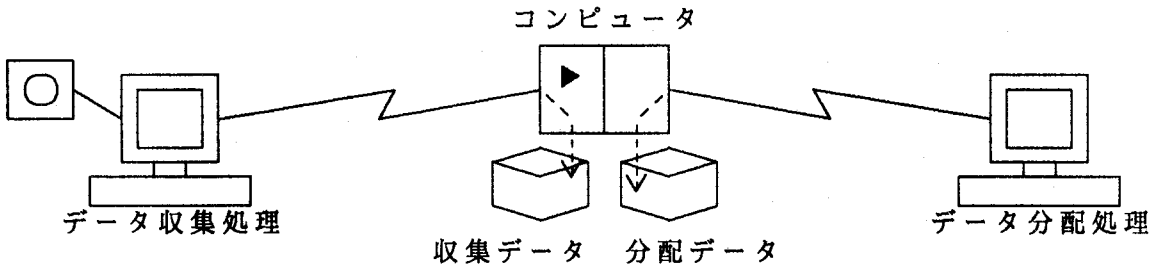
また、コンピュータ環境の大規模化、統合化、業務部門ニーズに合うシステム化の実現が求められつつある中で、情報処理技術者にとっての通信技術に関する基本的な知識、自らネットワーク環境を整備していく能力、また、普及されているネットワークアプリケーションをうまく情報の入手、伝達、交換などに利用する技能も求められている。

情報処理技能者養成施設においては、コンピュータネットワークのメリットを最大限に享受する環境が必ずしも整えられているわけではないだろうが、今後情報処理技術に関与するものとして、パソコン通信、施設内電子メールの交換、ニュースの利用は指導教官も率先して行うとともに、生徒にも習慣化するよう運用を図るべきである。また、世の中にどのようなコンピュータネットワーク形態があり、またどのようなネットワークアプリケーションが商用化され利用されているかを図で解説することが望ましい。

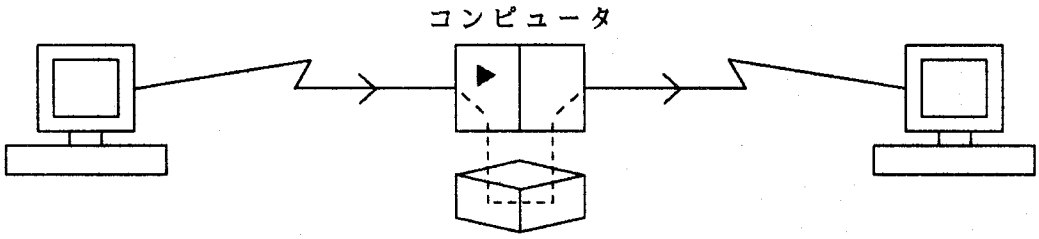
①　データベースとデータコミュニケーション



② データの集配処理



③ メッセージ交換処理



④ ＶＡＮ（付加価値網：［Value Added Network］）

電気通信業者が、通信回線を通して以下のようなサービスを提供する。

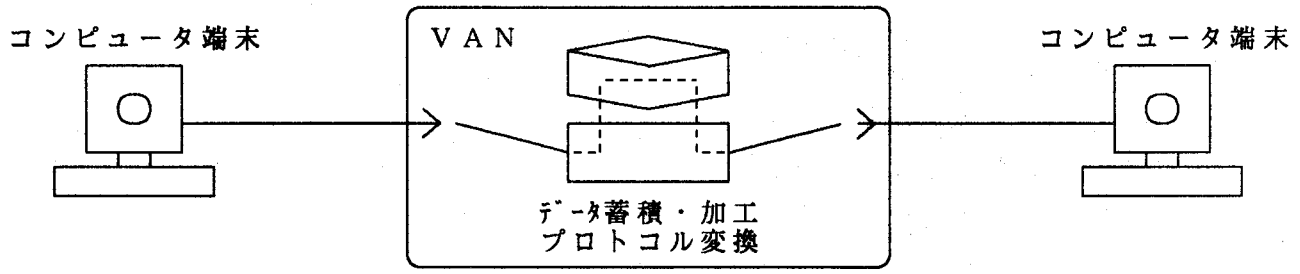
(a) 基本通信サービス　利用者に専用線、交換回線（回線交換、パケット交換）を提供。

(b) メールサービス　宛先に電文を転送。（パソコン通信、ＦＡＸ蓄積交換等）

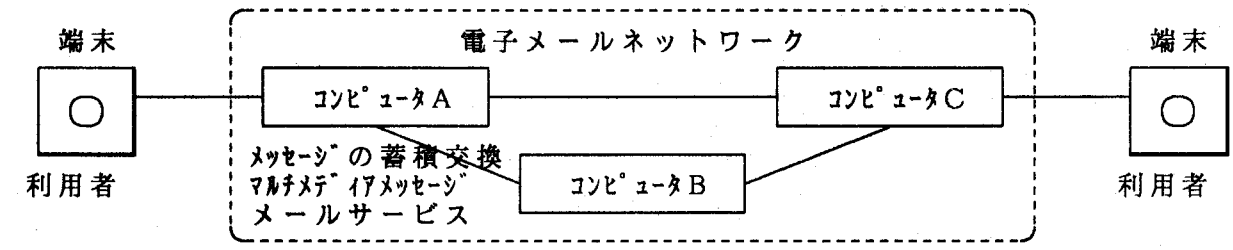
(c) ﾃﾞｰﾀ集配信ｻｰﾋﾞｽ　伝票受け渡しを電子化。（例：卸業と小売業の電子注文等）

(d) ﾘﾓｰﾄｺﾝﾋﾟｭｰﾃｨﾝｸﾞｻｰﾋﾞｽ　データの加工処理を代行。

(e) ﾃﾞｰﾀﾍﾞｰｽｻｰﾋﾞｽ　データベースを持ち、それへの参照サービスを提供。



⑤ 電子メールネットワーク



4．データ伝送

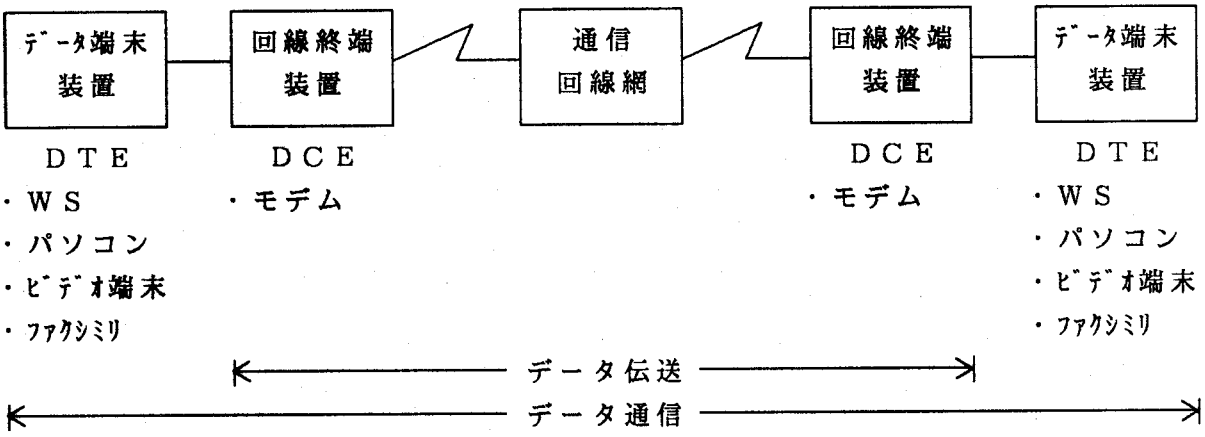
（１）データ通信、データ伝送

①　データ伝送

異なる地点間でのデータの伝送を実現するもので、通信回線やモデム等の装置が含まれる。

②　データ通信

データ伝送機能と、コンピュータや端末におけるデータの処理機能をも含むものをデータ通信という。



ＤＴＥ：Data Terminal Equipment

ＤＣＥ：Data Circuit terminating Equipment　（データ回線終端装置ともいう）

（２）通信回線の種類

①　アナログ回線

アナログ信号を伝送する回線。インタフェースは周波数帯域により規定される。

・音声帯域通信　－　３００Ｈz～３．４kHz。　（例：電話回線）

・広帯域通信　－　４８kHZ、２４０kHZなど

この回線では、ＤＴＥからのディジタル信号をアナログ信号に変換（変調：[modulation]）したり、変調されたアナログ信号をディジタル信号に逆変換（復調：[demodulation]）をする必要がある。これを行う装置をモデム（変復調装置：ＭＯＤＥＭ）という。

変調方式は、以下に代表される。

・周波数変調（ＦＭ：[Frequency Modulation]）

・振幅変調（ＡＭ：[Amplitude Modulation]）

・位相変調（ＰＭ：[Phase Modulation]）

---

② ディジタル回線

ディジタル信号を伝送する回線。インタフェースはビット速度により規定される。

③ 衛星通信回線

全地球レベルの長距離間での通信を可能としたもの。送信局から情報をアップリンク電波にのせて通信衛星へ送り、電波を増幅した後ダウンリンク電波に変換して受信局に送る無線通信である。

（３）伝送コード

データ伝送情報は、０と１の組み合わせで表現されるビット（bit）が基本となる。これを符号（コード）化という。このコードはＩＳＯやＣＣＩＴＴを中心に標準化が進んでおり、現在は７ビット単位の情報コードが制定されている。

我国では、情報交換用符号（ＪＩＳ規格Ｘ０２０１）により規定される７単位符号、８単位符号、および情報交換用符号系（ＪＩＳ規格Ｘ０２０８）により規定される１６ビット単位符号があり、目的に応じて使い分けがされている。

（４）伝送方式

データの伝送方式には、

・直列方式　－　符号を構成する各ビットを順次送る。
・並列方式　－　符号を構成するビットを並列に同時に送る。伝送速度は速くなるが、変復調回路が並列数必要となり、高価になる。

の２通りがあり、一般には直列方式が用いられる。

（５）伝送速度

伝送速度は、単位時間に伝送される情報の量で表される。この尺度は以下の通り。

・変調速度　－　(modulation rate)

１秒間に１要素を送る速度を１ボー（Baud）という。１要素が１０ミリ秒であれば変調速度は１÷０．０１　＝１００ボーである。この単位はアナログ伝送に対してのみ用いる。

・データ信号速度　－　(data signalling rate)

１秒あたりに伝送できるビット数により表し、これをビット／秒（ＢＰＳ：[Bit Per Second]）という。

・データ伝送速度　－　(transfer rate)

単位時間に伝送されるデータ量を伝送速度といい、字／分などが典型である。コンピュータネットワークで使われる専用回線、公衆通信回線での回線速度は、１２００、２４００、４８００、９６００ＢＰＳなどがある。

なお、“ボー”と“ＢＰＳ”とは常に同じ値であるとは限らないことに注意が必要である。

（６）通信方式

① 単方向通信 － (simplex)
データの伝送方向が一方向に固定されている。データの配信、データの収集専用に用いられることがある。現在ではほとんど使用されていない。

② 半二重通信 － (half duplex)
一時点では単方向にしかデータを伝送できないが、端末での切換によって逆方向からの伝送も可能である。現金引き出し機などはこのシステム例である。

③ 全二重通信 － (full duplex)
送信用、受信用両方の伝送路を持っているので、同時に両方向でデータを伝送できる。一般には二対の伝送路で送受信を行う４線式が多いが、一対の伝送路による２線式でもこの通信が可能である。

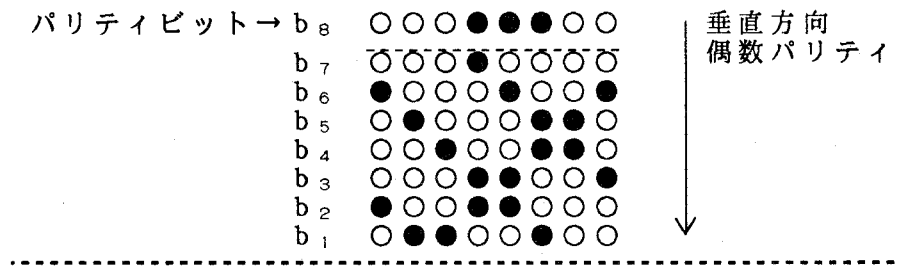
（７）誤り制御

データ伝送においては、伝送の誤りはしばしば発生するものであり、誤りの検出は非常に重要な要素である。データの誤りを検出したり、誤りに対して必要なら訂正を行うことを誤り制御という。

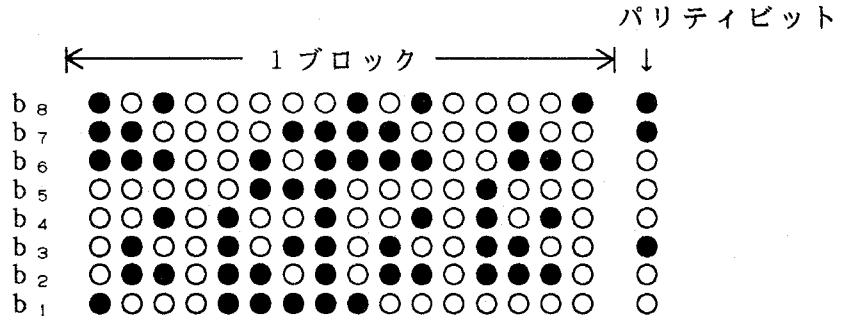
誤り制御方式には、
・垂直パリティ検査（ＶＰＣ：[Vertical Parity Check]）、水平パリティ検査（ＨＰＣ：[Horizontal Parity Check]）に代表される伝送データに冗長性をもたせ、送信側で誤り検出を可能とする情報を付加し、受信側で誤りを確認する方式。
・伝送の方法に冗長性を持たせる方式
の２種類がある。

このうち、前者の方式において検出される誤りをビット誤りと呼び、その発生確率をビット誤り率という。ビット誤り検出方法としては上記のようなパリティ検査と、サイクリックリダンダンシー検査（ＣＲＣ：[Cyclic Redundancy Check]）がある。

[垂直パリティ検査]



[水平パリティ検査]



[ＣＲＣ（巡回冗長）検査]

データ長が長くてもＣＲＣ符号の発生と検査が容易にできるため、コンピュータネットワークで広範囲に用いられている。

検査対象データのビット列を多項式Ｐ（ｘ）とし、検査用冗長符号（ＣＲＣ符号）の作成と誤り検査に使うビット列を生成多項式Ｇ（ｘ）とする。Ｐ（ｘ）の最上位のｘのべき数をｋとすると、ｋはＣＲＣ符号Ｒ（ｘ）の桁数となる。

＜送信側の処理＞

(a) $x^k \cdot P(x)$ {Ｐ（ｘ）をｋ桁分桁上げしたもの} をＧ（ｘ）で割る。割り算の余りのｋ桁をＲ（ｘ）とする。

(b) $F(x) = x^k \cdot P(x) + R(x)$ をＣＲＣ符号付きデータとして送信する。

＜受信側の処理＞

Ｆ（ｘ）をＧ（ｘ）で割る。このとき、Ｆ（ｘ）に誤りがなければＧ（ｘ）で割りきれるが、ビット誤りがあれば割りきれず、ビット誤りを検出できる。

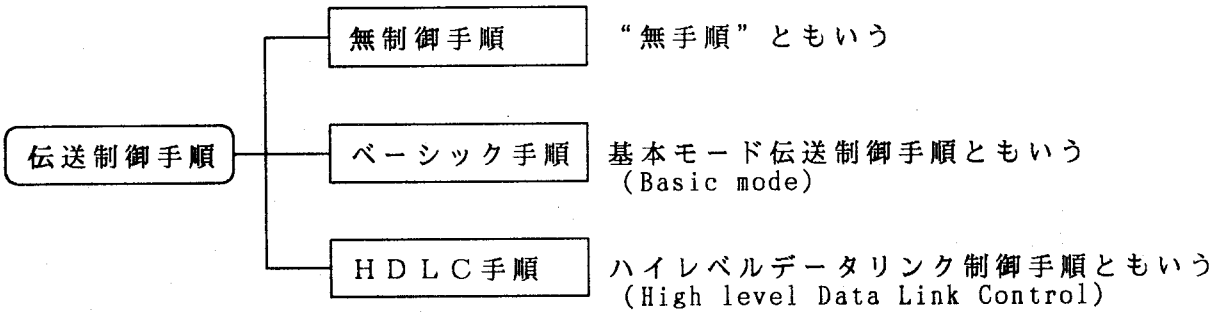
なお、国際標準ＨＤＬＣのＣＲＣ符号は１６ビットであり、生成多項式Ｇ（ｘ）は、

$G(x) = X^{16} + X^{12} + X^5 + X^0$ である。

5．伝送制御手順

コンピュータ同志、コンピュータと端末間で信頼度の高いデータ転送を行うために、手順が必要となる。これを実現するためにデータリンクというプロトコルが定められており、データの送受信に当たって、相手との接続の確認と接続を切るやりとりが必要になる。これを伝送制御手順という。また、データリンクとは、データの送受信のために張られた論理的な経路を指している。

伝送制御手順には次の種類がある。



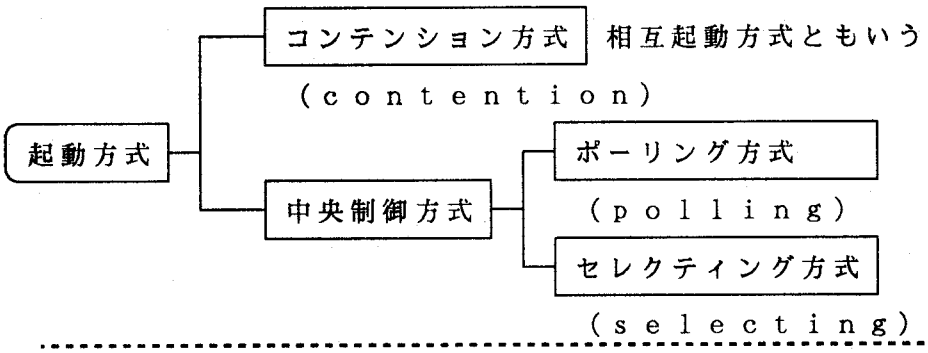
（１）無制御手順

端末装置の伝送制御による負荷を軽減し、端末を操作する利用者に対して誤りなどの確認を委ねる方式。伝送上の手順は決められてなく、端末からの入力データは１文字ずつ回線に送出され、またコンピュータからの受信データは端末にそのまま出力される。このケースでは、データの区切りを示すコード（文字）が予め相手との間で決められている必要がある。

（２）ベーシック手順

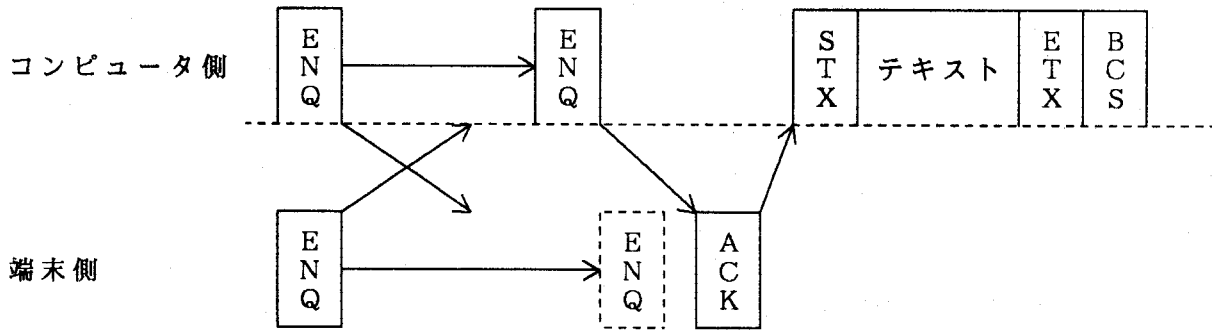
伝送制御符号［ＥＮＱ，ＡＣＫ，ＳＴＸ，ＥＴＸ，ＥＯＴなど］を用いた手順にそってデータ伝送を行う方式で、従来から広く用いいられている。

起動方式としては以下の種類がある。



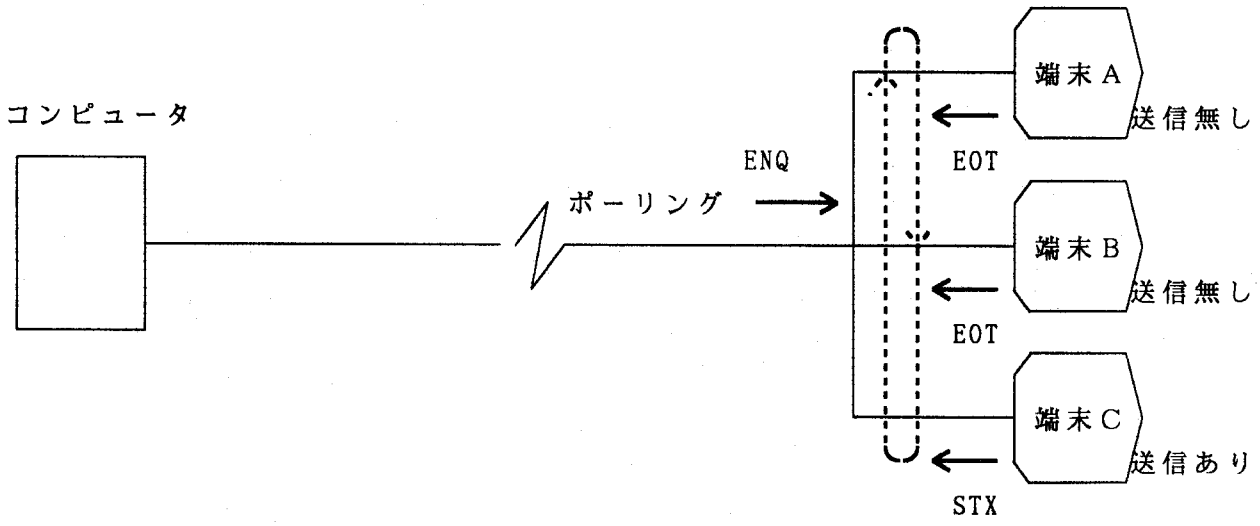
① コンテンション方式

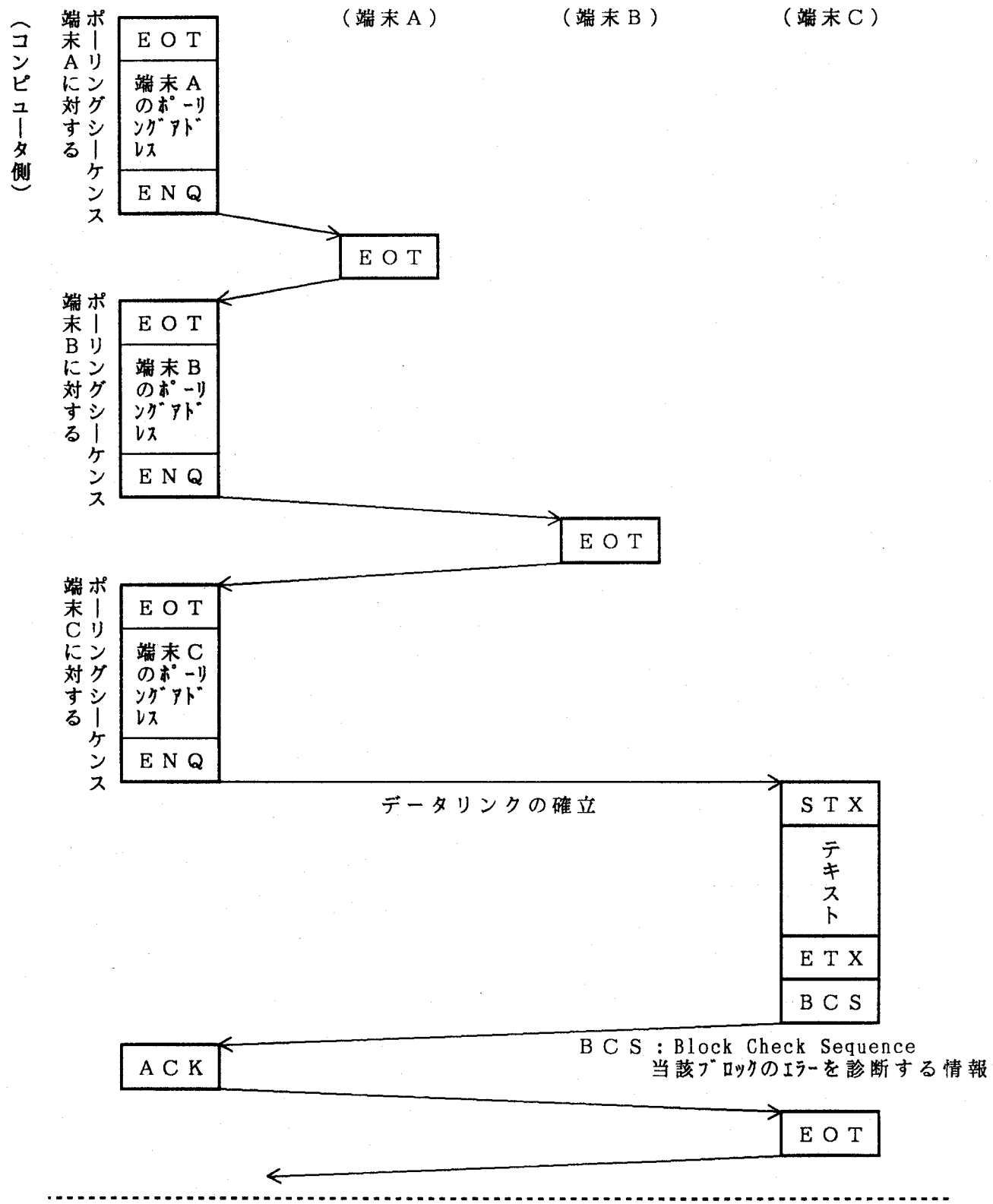
回線上でコンピュータ、端末どちら側からも起動（ＥＮＱ）できる。それぞれは一方的に相手に問い合わせできるため、互いの同時起動により衝突が発生し得る。この状態をコンテンション状態という。このような場合、互いに一定の間隔で再起動を行うといずれ「ＡＣＫ」が返送され、データリンクが確立される。



② ポーリング方式

コンピュータによる中央起動方式で、回線につながる端末からのメッセージ受信を司る。コンピュータは回線上の端末に対して一定間隔で順番に送信の要求を問い合わせるポーリングコードを送る。端末側は、送信の用意がある場合その旨の応答コードを送り、用意がなければテキスト終了コード（ＥＯＴ）を返す。

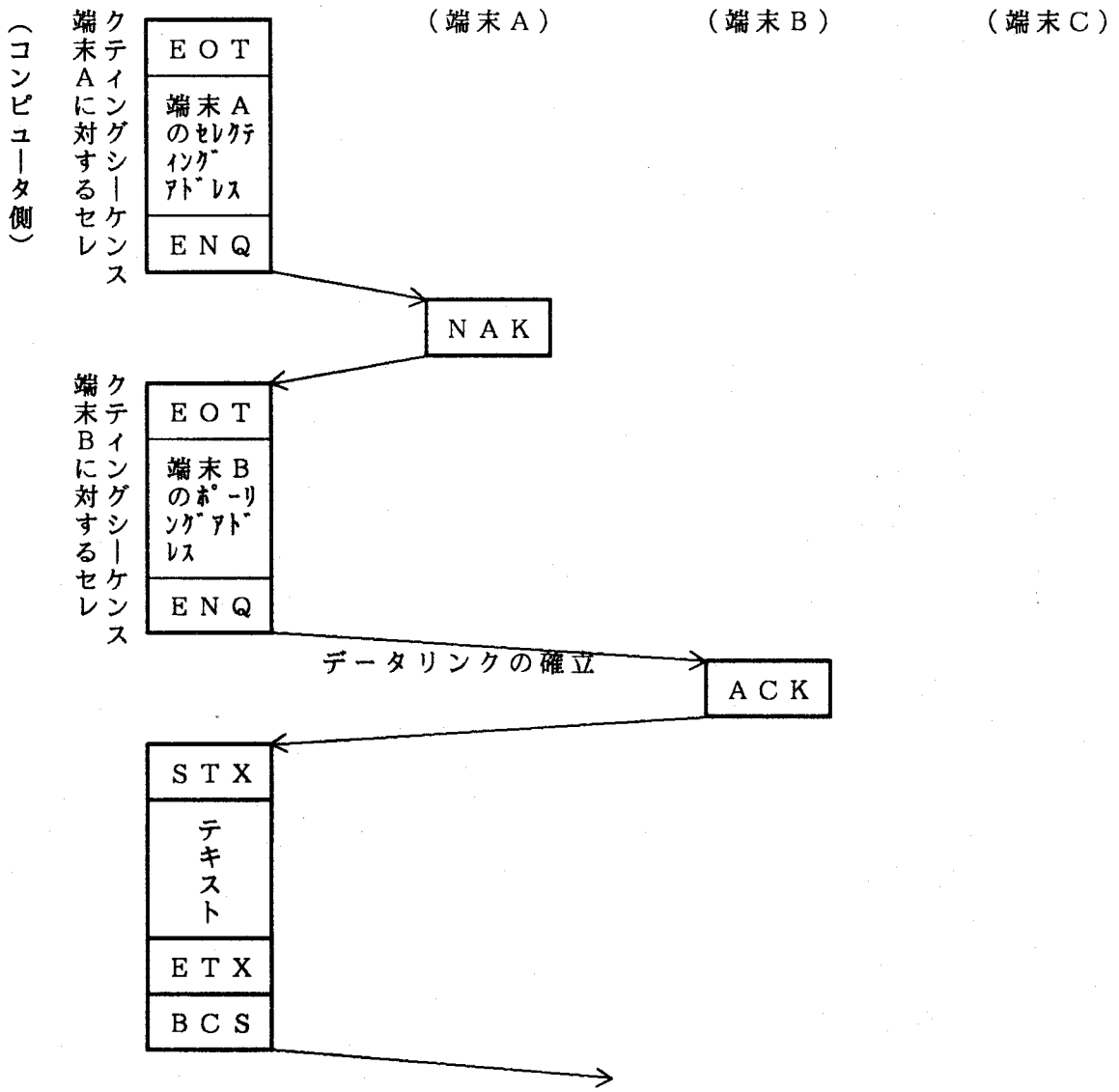



（コンピュータ側）
（端末Ａ）
（端末Ｂ）
（端末Ｃ）
ポーリングシーケンス
端末Ａに対する
ＥＯＴ
端末Ａのポーリングアドレス
ＥＮＱ
ＥＯＴ
ポーリングシーケンス
端末Ｂに対する
ＥＯＴ
端末Ｂのポーリングアドレス
ＥＮＱ
ＥＯＴ
ポーリングシーケンス
端末Ｃに対する
ＥＯＴ
端末Ｃのポーリングアドレス
ＥＮＱ
データリンクの確立
ＳＴＸ
テキスト
ＥＴＸ
ＢＣＳ
ＢＣＳ：Block Check Sequence
当該ブロックのエラーを診断する情報
ＡＣＫ
ＥＯＴ

③　セレクティング方式

コンピュータから端末に対してメッセージを送信する時、送信用コードを端末に送り端末側が受信可能であるかを確認する。端末側は、受信可能な場合ＡＣＫコードを返送し、コンピュータからのメッセージを待ち、受信できない状態であればＮＡＫコードを返す。

下図では、コンピュータから端末ＡとＢにメッセージを送信することを想定している。Ａは受信可能でなく、Ｂは可能であるものとする。



≪ベーシック手順における伝送制御符号≫

コード	名称		意味
ＳＯＨ	start of heading	ヘディング開始	情報電文のヘディングの開始
ＳＴＸ	start of text	テキスト開始	ﾃｷｽﾄの先行またはﾍﾃﾞｨﾝｸﾞの終結
ＥＴＸ	end of text	テキスト終結	テキストの終結を示す
ＥＯＴ	end of transmission	伝送終了	１つ以上のテキストの伝送終了
ＥＮＱ	enquiry	問い合わせ	相手からの応答を要求する
ＡＣＫ	acknowledge	肯定応答	送信側に対する受信側の肯定応答
ＤＬＥ	data link escape	伝送制御拡張	後続の文字の意味を変更する
ＮＡＫ	negative acknowledge	否定応答	送信側に対する受信側の否定応答
ＳＹＮ	synchronous idle	同期信号	端末側の同期をとり維持する
ＥＴＢ	end of transmission block	伝送ﾌﾞﾛｯｸ終結	分割されたﾃﾞｰﾀﾌﾞﾛｯｸの終わり

（３）ＨＤＬＣ手順

ベーシック手順に各種の機能を追加し、いくつものコンピュータを接続したようなデータリンクに対しても、効率よく、信頼性の高い伝送制御手順を実現したものとしてＨＤＬＣ（高水準データリンク制御）がある。

この手順では、フレームという単位で情報の伝送を行う。

1ﾊﾞｲﾄ	1ﾊﾞｲﾄ	1ﾊﾞｲﾄ	nバイト	2バイト	1ﾊﾞｲﾄ
フラグシーケンス	ｱﾄﾞﾚｽﾌｨｰﾙﾄﾞ	制御ﾌｨｰﾙﾄﾞ	情報フィールド（可変長）	フレーム検査シーケンス	フラグシーケンス

情報ﾌﾚｰﾑ➡ b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8
監視ﾌﾚｰﾑ➡ 0 N(S) P/F N(R)
1 0 S P/F N(R)
非番号 ➡ 0 1 M P/F M
ﾌﾚｰﾑ

＞受信局アドレス、送信局アドレス
＞１つのフレームの開始と終了

＞ＦＣＳ：Frame Check Sequence 誤り検出のためのﾋﾞｯﾄ列

(注)制御ﾌｨｰﾙﾄﾞの詳細についてはX.25のLAPB手順を参照のこと

（４）プロトコル

コンピュータとコンピュータ、コンピュータと端末間でメッセージの交換を行う場合、通信上の動作規約に従う必要がある。この規約がないと不特定多数のマシン間での適切なデータ送受信は不可能である。

プロトコルにはハードウェアレベルでの規定と、オペレーティングシステムレベルでの通信データの送受に関する規定、さらにはアプリケーションプログラム間の高度なやりとり（例えばファイルの転送など）についての規約が必要となる。

このプロトコルに関しては、単一のコンピュータや機器メーカの製品同志では規約が同じであるため相互接続が容易であったが、オープンシステム時代を迎え、マルチベンダ環境のコンピュータシステム構成をとるようになっており、異なるメーカの製品間での統一された標準的な規約が必要になってきた。

現在の処、ＯＳＩ（開放型システム間接続［Open Systems Interconnection］）の参照モデルが標準プロトコルとして広く普及しつつある。

≪ＯＳＩ参照モデルの各層の機能≫

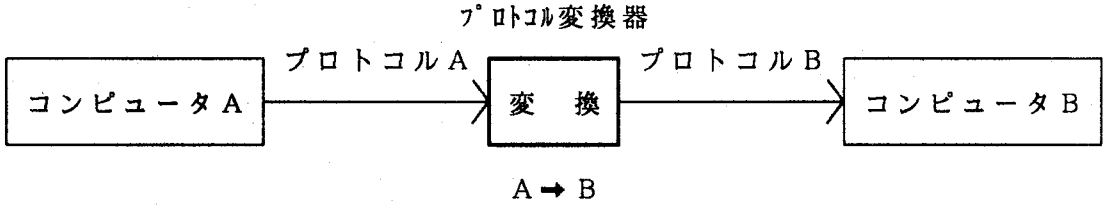
レイヤ	機能概要	電話の例
第７層（応用層）	応用プロセスや端末の利用者にデータ通信機能を提供する。メッセージ（文書）転送、ファイル転送、データベースアクセス等の業務に合うプロトコルを実現する。	共通の言葉での会話を行う
第６層（ﾌﾟﾚｾﾞﾝﾃｰｼｮﾝ層）	応用プロセスが、通信相手とのデータの型や符号、構造の違いを意識しないですむよう、共通構文に変換、逆変換を行う。	会話を行う言葉の確認を行う
第５層（セッション層）	応用プロセスの情報の送り方を決める。全二重通信、半二重通信、データの受取確認（送信権、同期、再送機能等）に関するコントロールを行う。	もしもしと相手を確認する
第４層（トランスポート層）	中継する通信網の品質に応じてデータ伝送の信頼性をコントロールする。また、ネットワークの利用者を意識するトランスポート・アドレスを管理する。	通話を保証
第３層（ネットワーク層）	エンド・システム間の論理的な通信路を提供する。通信ルートを選定し、データを中継、転送して送信する。相手番号（アドレス情報）による通信路が設定される。	相手番号をダイヤルし相手と接続
第２層（データリンク層）	隣接するノード間の通信路を提供し、フレーム単位でのデータ伝送を保証する。ＷＡＮでは、ＨＤＬＣが、ＬＡＮではＭＡＣの伝送制御手順が対応する。	受話器を上げ電話網と接続する
第１層（物理層）	物理的、電気的条件を管理し、ビット単位の伝送を保証する。	電話をコンセントに繋ぐ

異なるプロトコルを持つコンピュータ間、コンピュータと端末間でメッセージの交換を行う場合には、どちらかの側がプロトコルを合わせる必要がある。即ち自マシン（または端末）でのプロトコルを他マシン（または端末）のプロトコルに変換しなければならない。この操作をプロトコル変換という。

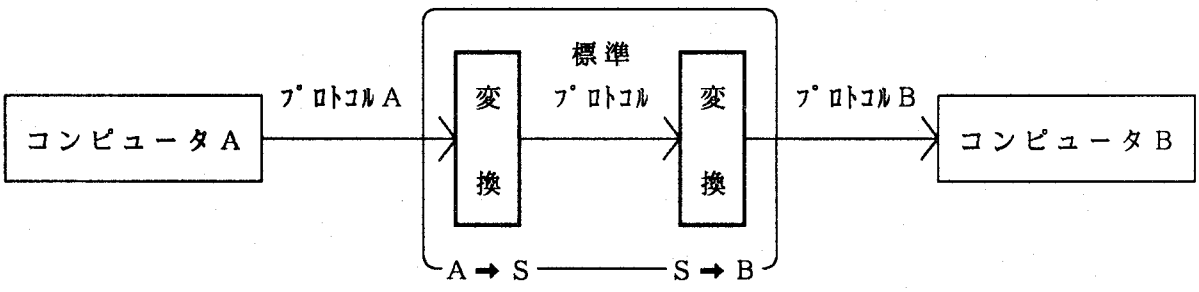
これは、異なるプロトコルを持つコンピュータ間での相互通信を可能とするための仕掛をハードウェアまたはソフトウェアで実現するものである。

以下に、プロトコルの変換例を示す。

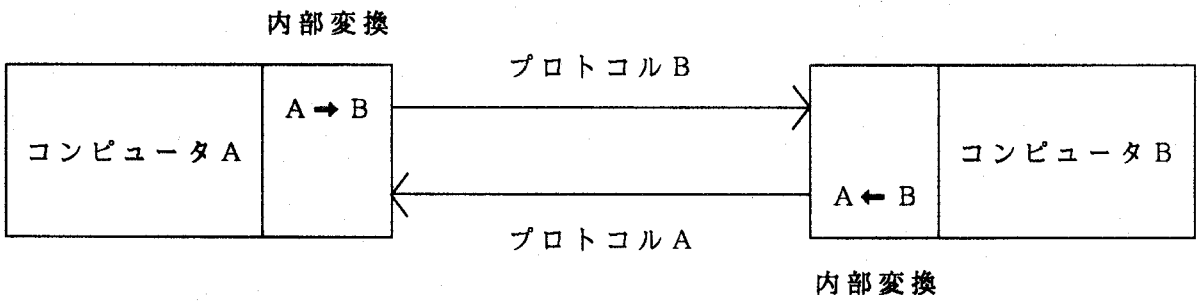
① プロトコル変換器による変換



② 付加価値回線網内の変換機能による



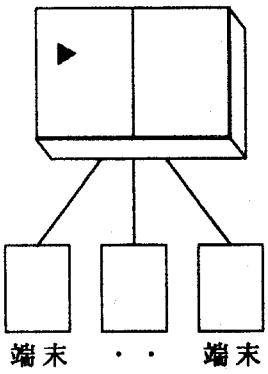
③ ソフトウェアによる変換（上位レイヤで）



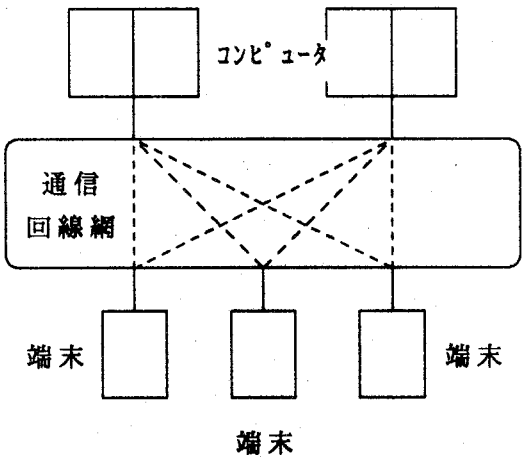
6．通信ネットワーク技術の利用

（１）ネットワークを介したコンピュータ処理

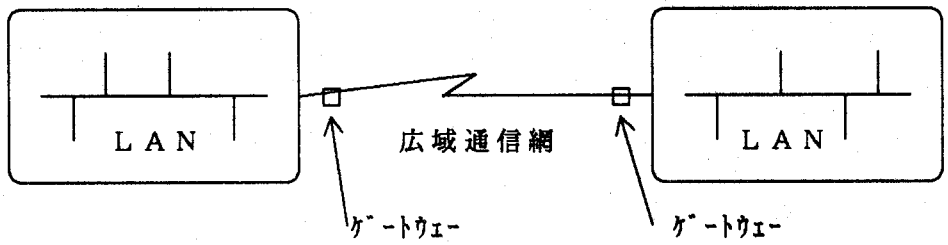
① 専用オンライン型



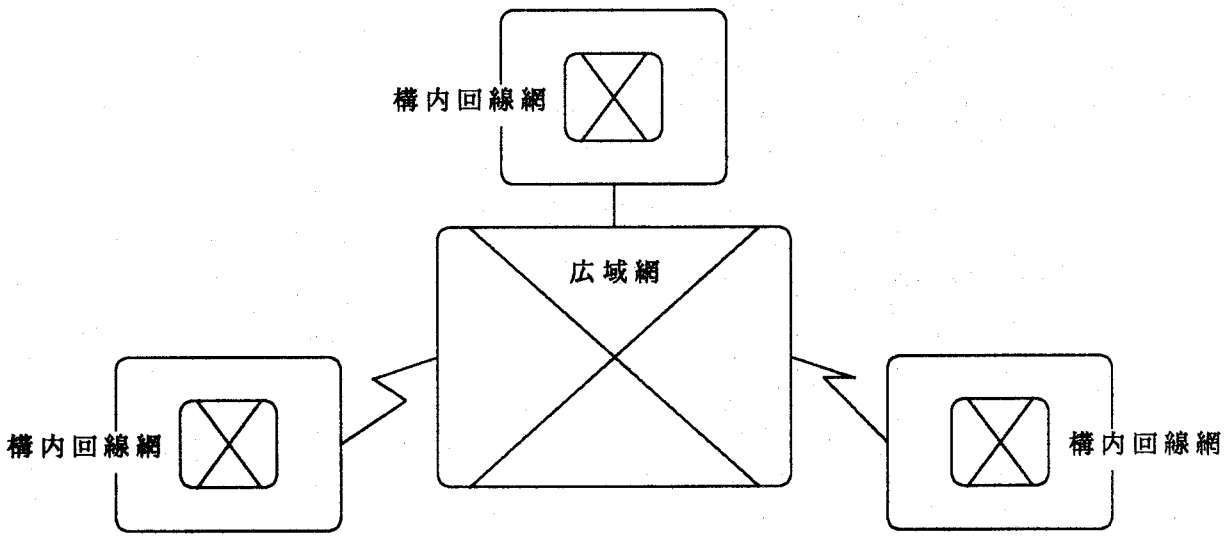
② 広域オンライン型



③ ネットワーク相互接続型

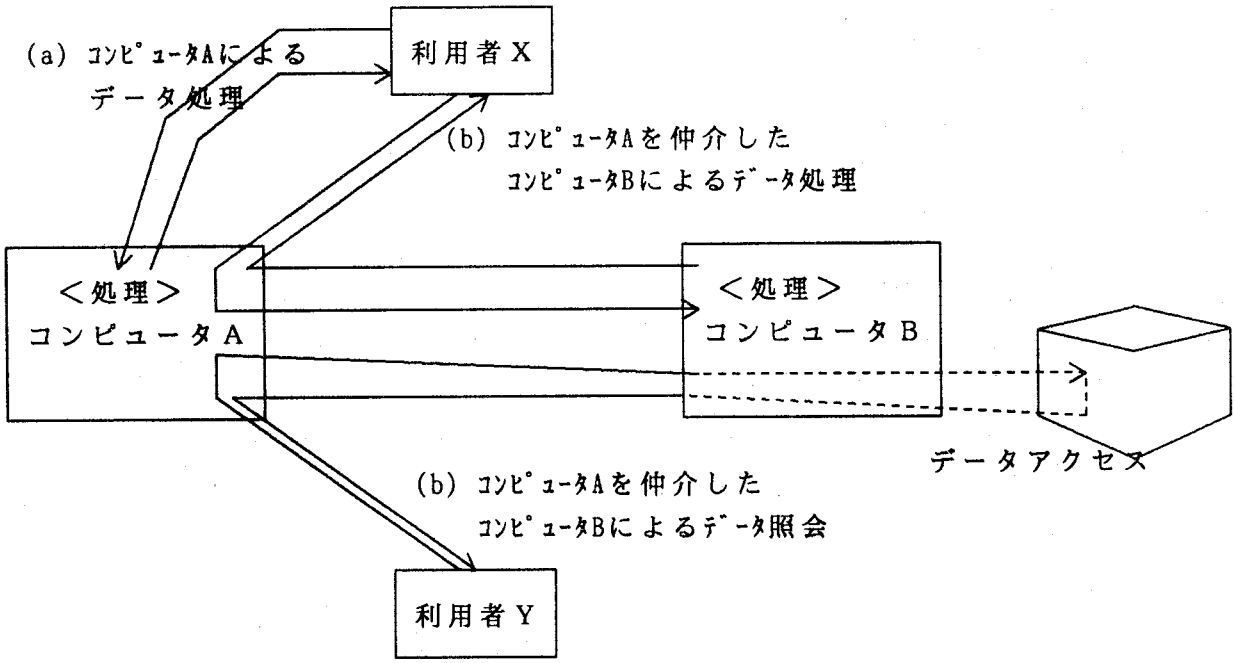


④ 構内網＋広域網型

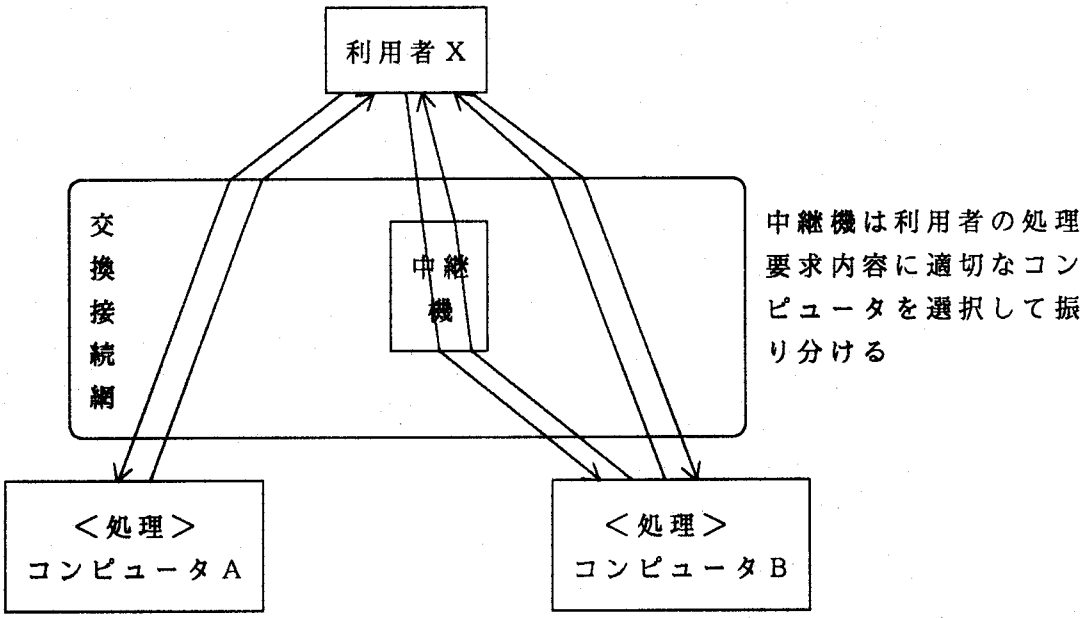


（２）ネットワークを介したコンピュータ処理の流れ

① 直接ジョブ処理、ジョブ転送処理、ファイル転送処理



② 交換接続網

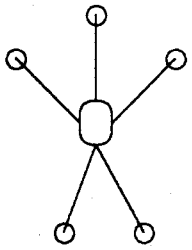
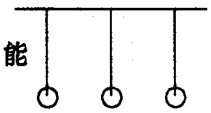
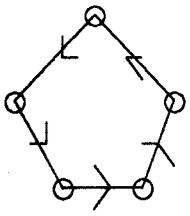


（３）ローカルエリアネットワーク（ＬＡＮ：[Local Area Network]）

ＬＡＮ技術の代表的な応用分野はやはりＯＡ（Office Automation）が主であり、その他にもＦＡにおいても活発である。特にＯＡにおいては、組織内における情報交換、情報流通のための情報伝達、通達文書の発信、離れた居室間での電子会議などの強力なツールとして広く活用されている。

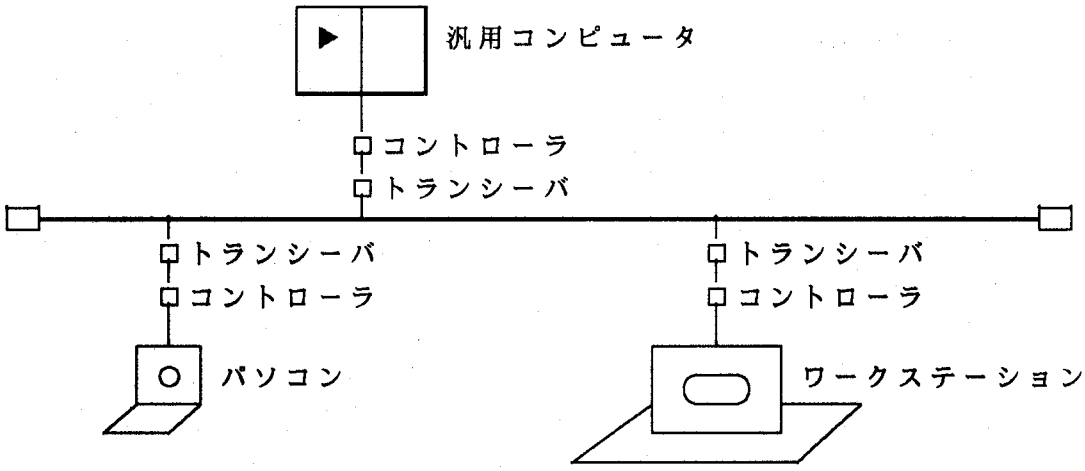
①　ネットワークトポロジー

ＬＡＮは伝送媒体とトポロジーにより分類ができる。以下で、それぞれの特徴を整理する。

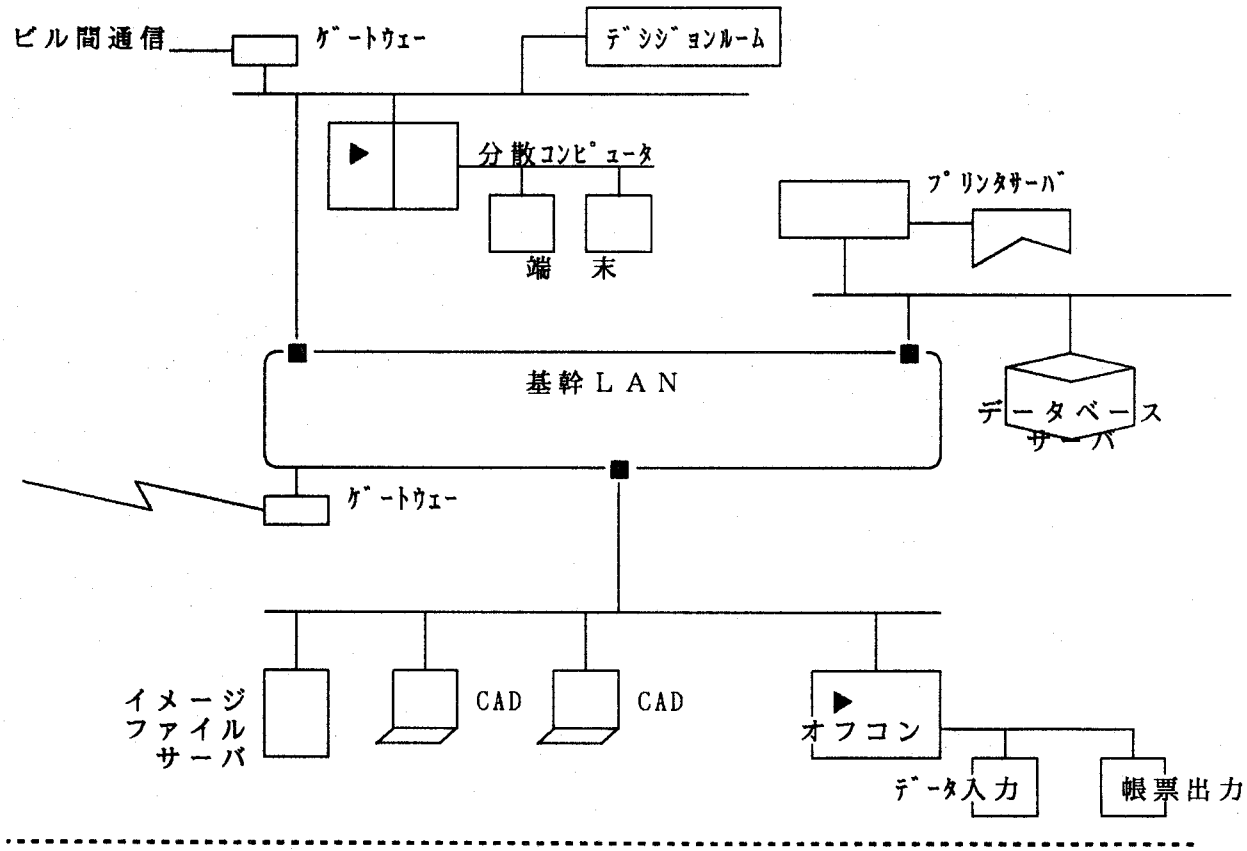
分類	機能的な特徴
スター型	低速な情報通信に適する 通常、中央コントローラとしてコンピュータやＰＢＸを使用 中央コントローラの障害により全体がダウンする よりつい線などを利用でき安価にシステムを作れる 比較的大規模なシステムを構築でき、接続コストが低減する ワークステーション、パソコン、また電話機やＦＡＸを繋ぐ 
バス型	高速な通信を実現できる 最も広く普及しており、小規模から比較的大規模構成が可能 負荷が小さいときに応答がよい 送信先への到着確認はソフトウェアで行う 各ノードの故障は全体には影響しない ノードとしてはワークステーション、パソコンが多い 
リング型 ループ型	高速な通信を実現できる リング状に情報が一方向に巡回する 長距離で大規模なＬＡＮを構築できる 高負荷にも耐えられるが、コスト高となる リング型では、送信権制御をノード側が対等な関係で行う ループ型では、コントローラが存在し、全体の制御を行っている ノードの故障が全体に波及する。この障害対策に工夫が必要 

② ＬＡＮの構成例

(a) 光ファイバケーブルなどによるコンピュータ同志の接続



(b) オフィスオートメーション事例



指導上の留意点

通信ネットワーク技術によって情報処理の世界が急激に広がることを最大の理解事項とする。また、通信ネットワーク技術は、ネットワークに関係するハードウェアのみでなく、端末間、コンピュータ間、コンピューター端末間で色々な約束ごとがあって初めて成立することを理解させることが重要である。

この約束ごとに従ってネットワークにおけるノード間で行われる処理は基本的にソフトウェアが行っていることを強く認識させる必要がある。

技術的な面で深入りする必要はなく、原理を理解させることが第一である。

用語

バッチ処理、一括処理、ローカルバッチ処理、センターバッチ処理、リモートバッチ処理、
オフライン処理、
タイムシェアリング処理、会話型処理、対話処理、
オンラインリアルタイム処理、即時処理、実時間処理、オンライン処理、
ローカル処理、遠隔処理、リモート処理、
リモートジョブエントリ、ＲＪＥ、タイムシェアリングシステムローカル処理　　対話型
クライアント／サーバ、フロントエンド処理、バックエンド処理、
ファクトリーオートメーション、

シンプレックスシステム、デュプレックスシステム、デュアルシステム、
マルチプロセッシングシステム、タンデムシステム、分散処理システム、

データコミュニケーション、データの集配信、メッセージ交換、ＶＡＮ、付加価値網、

データ伝送、データ通信、ＤＴＥ、ＤＣＥ、
アナログ回線、モデム、ディジタル回線、衛星通信回線、伝送コード、伝送方式、
ボー、ＢＰＳ、
単方向通信、半二重通信、全二重通信、誤り制御、垂直パリティ検査、水平パリティ検査、
ＣＲＣ、巡回冗長検査、
伝送制御手順、無制御手順、ベーシック手順、基本モード伝送制御手順、
コンテンション方式、ポーリング方式、セレクティング方式、
ＨＤＬＣ手順、ハイレベルデータリンク制御手順、
プロトコル、プロトコル変換、ＯＳＩ参照モデル、開放型システム間接続、

ＬＡＮ、ローカルエリアネットワーク、スター型、バス型、リング型、ループ型、
オフィスオートメーション

第2種情報処理技術者試験

○　情報処理システムに関連する知識
- システムの処理形態（オンライン処理、バッチ処理など）に関すること。
- システム構成（デュプレックスシステム、マルチプロセシングシステムなど）に関すること。
- システムの性能評価（応答時間、アクセス時間など）に関すること。
- その他（ＯＡシステム、ＦＡシステムなど）

○　伝送制御手順に関連する知識
- 伝送制御手順（ＨＤＬＣ手順、基本形データ伝送制御手順など）
- プロトコル
- 伝送速度
- 誤り制御

第８章　ソフトウェア全般に関連する知識

内容のあらまし

内　　容	説　明	議　論	机上実習	計算機実習
システムの性能について	コンピュータシステムの評価に関する考え方を説明する。 応答時間、スループット、性能改善のポイントを理解させる。	応答時間やスループット改善のために、どのような手だてが必要かを議論させる。	ソフトウェアにつき理解した範囲で、アプリケーションプログラムの性能改善策を抽出させる。	特になし
ＲＡＳＩＳ、信頼性の確保について	ＲＡＳＩＳについて、それぞれの確保の意義、評価指標について理解させる。 MTBF、MTTRの計算法は必須。	コンピュータ社会の進展において、ＲＡＳＩＳ、信頼性の確保がどのような意義を持つかについて議論させる	ＭＴＢＦ、ＭＴＴＲ、稼働率について練習問題を解く。	
決定表	決定表の有用性、表の形式、表の作り方、決定表とフローチャートの関係について理解させる	決定表の有利な点、アルゴリズムなどの説明にどう生かすかを議論させる	第２種情報処理技術者試験からの問題など	
コード設計、帳票設計	コードの目的、コードの機能、コード化の手順、および、帳票レイアウト作成方法について理解させる。	プログラム開発における、コード設計、帳票設計がどのように重要であるかを理解させる。	小さなシステム化事例におけるコード、および帳票の設計について練習する	

指導内容

1．システムの性能について

システム性能の評価、確保については、次の３つの観点から検討する必要がある。

（１）コンピュータシステムの性能評価

まず、コンピュータシステムの性能に関して全体的な目標をどう設定するかを明確にし、それに基づき、ジョブやタスクの優先順位をどう設定すべきかを決める必要がある。

一般に汎用コンピュータにおける性能向上の２大ポイントとして、

○　応答時間（ターンアラウンドタイム：Turnaround Time ）の最小化

○　単位時間処理量（スループット：Throughput）の最大化

が挙げられる。前者では、オンラインリアルタイム処理系の応答性をどう重要視するか、また、後者では、バッチ処理を含めジョブのトータルな処理量にどう重きを置くかに関連している。

この応答時間の高速化要求と、単位時間処理量の増大は裏腹の関係にあり、一方を強調し過ぎると他方の評価が一気に落ちることになる。そのバランスをどうとるかが重要なことであり、結局どのような性能の確保に重点を置くかにかかってくる。

目標の設定	性能の確保策	他　へ　の　影　響
応　答　時　間 の最小化	優先度の高いトランザクションにＣＰＵ、入出力装置その他の資源をできるだけ占有させる	資源アイドルタイムが増加する
単位時間処理量の 最　大　化	多数のバッチジョブ、トランザクションジョブを同時並行処理させる	資源の使用効率が上がるが、多重性が大きすぎると資源待ちが増えるため応答性が低下する

例えば、応答時間を向上させるために、単一のトランザクションに中央コンピュータ資源を多く割り当てる方法だけでなく、端末側のインテリジェンスやローカル処理性を高めることにより、中央コンピュータの負荷を減らし、応答時間と、スループットの向上を図る手がある。

（２）システム開発概要設計における性能の目標設定

概要設計においては、少なくとも以下の事項について綿密な調査を必要とする。

性能設計	性能設計に際しての留意事項
性能目標値の明確な設定	○　応答時間 ○　資源の使用量 ［主記憶サイズ、２次記憶サイズ、入出力回数、チェネルへの負荷、回線の使用量など］
性能目標の承認	利用者からの、設定した性能目標の了解
性能目標設定条件の妥当性	○　ハードウェア環境　－　主として機器の性能 ［平均命令実行時間、チェネル転送速度、入出力速度、主記憶容量、２次記憶容量、回線速度など］ ○　ソフトウェア　－　ＯＳやメーカより提供されるプログラム ○　データの条件 ［平均データ量、ピークデータ量など］

なお、開発しようとしたり、あるいは基幹とはなるが購入を予定しているプログラムについてどの程度の性能が出るかを予測する各種のベンチマークテスト手法がある。

（３）システム運用時の性能改善

	チューニングの必要性	留　意　点
システム性能の監視と改善	○適切なファイル占有量、適切なアクセス量 ○装置への偏りのないアクセス ○ＯＳサービスの負荷状況 ○バッチジョブのターンアラウンド ○計算量の多いジョブへのサービス	○実装主メモリの量、スワップ領域の量、ページング率 ○キャッシュメモリの大きさ ○入出力装置の設置台数 ○ＣＰＵ能力の適切性
アプリケーションの性能改善	○適切な入出力回数とブロック長 ○必要な主メモリの制御 －不必要なメモリ確保の回避 －メモリ内処理に必要な量の確保 ○冗長なコーディングの最適化	○データの項目、データ処理目的にあったファイル編成の採用を確認 ○メモリ処理、ファイル処理の適切な場合分け ○必要以上の資源独占の回避

2．ＲＡＳＩＳ、信頼性の確保について

コンピュータシステム、ネットワークシステムにおける障害に対する予防、発生後の早期回復、信頼性をどう確保するか、また、可用性を高めるために稼働率をどう計算するかは、コンピュータのシステム構成、ネットワーク構成を決める上で非常に重要なポイントとなる。

システムにはＲＡＳＩＳという能力が要求され、これはハードウェア、オペレーティングシステム、アプリケーションそれぞれが役割を担いながら実現されている。

即ち、情報通信システムの優良性を評価する尺度としてＲＡＳＩＳがある。これは機能の多様性、大量な処理性、性能等の能力の拡大ばかりでなく、システムの安全性、システムに対する各種の脅威からの保護性も有していなければならないことを示している。

ＲＡＳＩＳ
- Ｒ（信頼性［Reliability］）－ 故障なしに処理し続けられること ＜指標：ＭＴＢＦ＞
- Ａ（可用性［Availability］）－ システムがいつでも正常な状態にあること ＜指標：稼働率＞
- Ｓ（保守性［Serviceability］）－ システムに障害が起こっても容易に回復できること＜指標：ＭＴＴＲ＞
- Ｉ（完全性［Integrity］）－ システムやデータの破壊に対して復原力があること
- Ｓ（機密性［Security］）－ データの外部漏洩、改竄、紛失を防止できること

① 信頼性

コンピュータやネットワークシステムが故障なく使えるかどうかを測る指標。
故障が直ってから、次に故障が起こるまで連続して使える平均時間のことで、この値が大きい方が信頼性は高いことを示す。勿論ソフトウェアに対しても同じ指標となり得る。

この場合、故障の発生頻度のことを“平均故障間隔（ＭＴＢＦ［Mean Time Between Failure］）”といい、以下の式で表す。

$$\text{平均故障間隔} = \frac{\text{総稼働時間}}{\text{故障回数}}$$

② 可用性

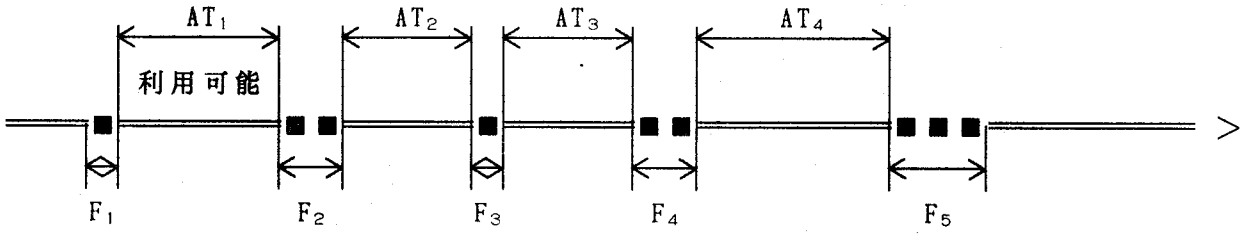
可用とはコンピュータ（ハード、ソフト含め）やネットワークがいつでも使える状態にあることをいう。すなわち、システムがどのくらいの時間正常に動作しているかを示すものである。これを“稼働率”といい、システムが正常動作をしている確率を意味しており、以下の式で表せる。

$$\text{稼働率} = \frac{\text{利用可能時間}}{\text{合計時間}}$$

③ 保守性

コンピュータやネットワークシステムが故障したときに修復にどの程度の時間がかかるかを表すのに、“平均修復時間（ＭＴＴＲ［Mean Time To Repare］）”が用いられる。これは故障発生時点から修復が終わるまでの平均時間を表し、修復にかかった時間の平均ということになる。この値が小さいほど保守性が高いことを示す。

ここで、ＭＴＢＦ，ＭＴＴＲおよび稼働率間の関係について整理しておく。



例えば $\frac{AT_1+AT_2+AT_3+AT_4}{4}$ がＭＴＢＦ、 $\frac{FT_1+FT_2+FT_3+FT_4}{4}$ がＭＴＴＲを表している。

また、稼働率 $= \frac{\text{ＭＴＢＦ}}{\text{ＭＴＢＦ＋ＭＴＴＲ}}$ の関係が得られる。

④ 完全性

データの人為的（故意または誤操作など）、あるいは自然災害などにより破壊された場合に、影響を最小限にして破壊される以前の状態に復元できるときに完全性が保たれる。完全性を高く保つために、ファイルの多重化、バックアップの採取が必要となる。

⑤　機密性

ハードウェア的な障害、およびデータに対する不法なアクセスに対してどのくらいの保護を行うかを表す。

例えば、利用者に対するパスワードの設定、アクセス権限の制限、また、オンラインシステムにおけるデータの暗号化が行われている。

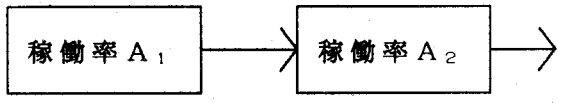
⑥　稼働率の計算

稼働率は、システムが稼働している確率を表すことから、故障率は１－稼働率となる。稼働率は、システムの信頼性の指標として用いられるが、これは機器の結合形態により変化する。

稼働率A_1、A_2の機器があった場合に、

直列システムでの全体の稼働率は、$A_1 \times A_2$

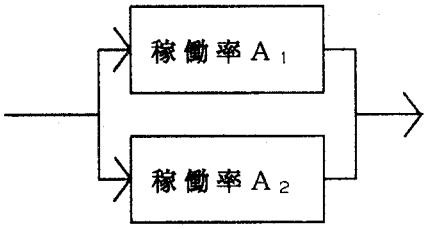
（いずれか一方の故障でシステム全体が故障）



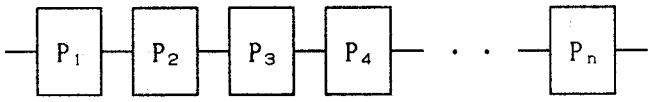
並列システムでの全体の稼働率は、

$$A = 1 - (1 - A_1) \times (1 - A_2)$$
$$= A_1 + A_2 - A_1 \cdot A_2$$

（注）$(1 - A_1) \times (1 - A_2)$は機器１、２共に故障する確率を表す

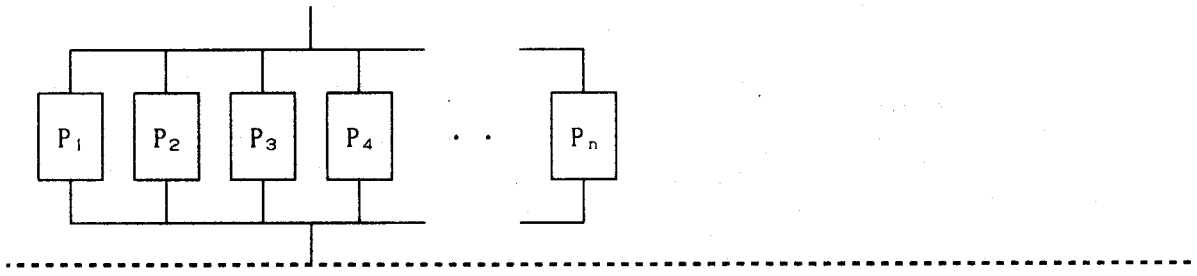


一般に、ｎ個の機器の直列構成のシステム全体の稼働率は、$P_1 \times P_2 \times P_3 \times \cdots \times P_n$で表される。



また、ｎ個の機器の並列構成のシステム全体の稼働率は、

$1 - (1 - P_1) \times (1 - P_2) \times (1 - P_3) \times \cdots \times (1 - P_n)$で表される。

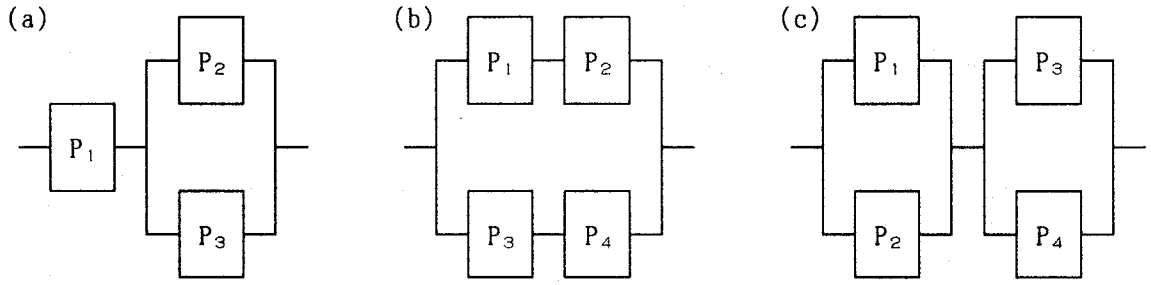


直列、並列混合型の構成における全体の稼働率もこれらの延長で計算できる。

（a）$P_1 \times \{1 - (1 - P_2) \times (1 - P_3)\}$

（b）$1 - (1 - P_1 \times P_2) \times (1 - P_3 \times P_4)$

（c）$\{1 - (1 - P_1) \times (1 - P_2)\} \times \{1 - (1 - P_3) \times (1 - P_4)\}$



3．決定表（デシジョンテーブル［decision table］）

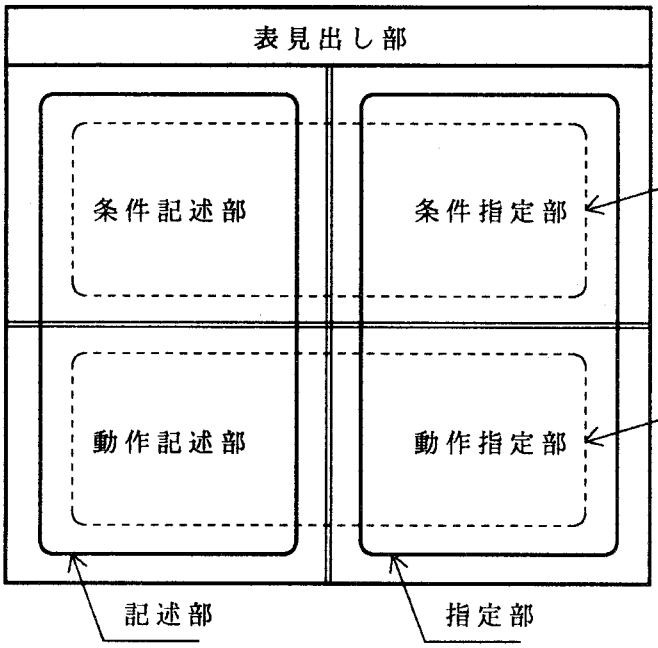
決定表は、いくつかの条件について満足する状況と、それに対応すべき行動（あるいは作業）を簡潔に表現する機能を持つ。

条件が入りくんでいる場合に、フローチャートのように処理の流れとして図示するとかえって全体的に何をしようとしているのかわかりにくくなりがちである。

決定表を用いると、話の筋が簡潔明瞭に整理され、誰にも直感的に理解できる利点がある。

① 決定表の形式

図 一般形式と名称 その１



［用語解説］

(1) **決定表** 問題の記述において起こり得る全ての条件と、それに対して実行すべき動作を組み合わせた表。

(2) **単適合決定表** 条件の組合せのどの一つも、ただ一つの規則においてだけ満たされる決定表。

(3) **多重適合決定表** 条件の組合せの内の少なくとも一つが、二つ以上の規則によって満たされる決定表。

(4) **規則** 表の条件指定部及び動作指定部を通る一つの列であって、満たされる条件のただ一つの組合せと、それに対応する動作との組みを定義するもの。ある規則が満たされるとは、全ての条件がその規則の条件指定に合致することをいう。

(5) **補完規則（ELSE-rule）** 表中に値による指定が示されていない条件の全ての組合わせに対して、実行すべき動作を示す規則。

(6) **条件（condition）** 問題の表記において考慮すべき事項の記述または条件の一部として考慮すべき他の手続きへの参照。

(7) **動作（action）** 問題解決のために実行すべき処理の記述。

(8) **条件指定（condition entry）** ある条件と特定の規則との関連付け。

(9) **動作指定（action entry）** ある動作と特定の規則との関連付け。

(10) **条件記述部（condition stub）** 問題の記述において考慮すべき全ての条件を列挙したもの。

---

(11)**動作記述部 (action stub)** 問題の記述において実行すべき全ての動作を列挙したもの。

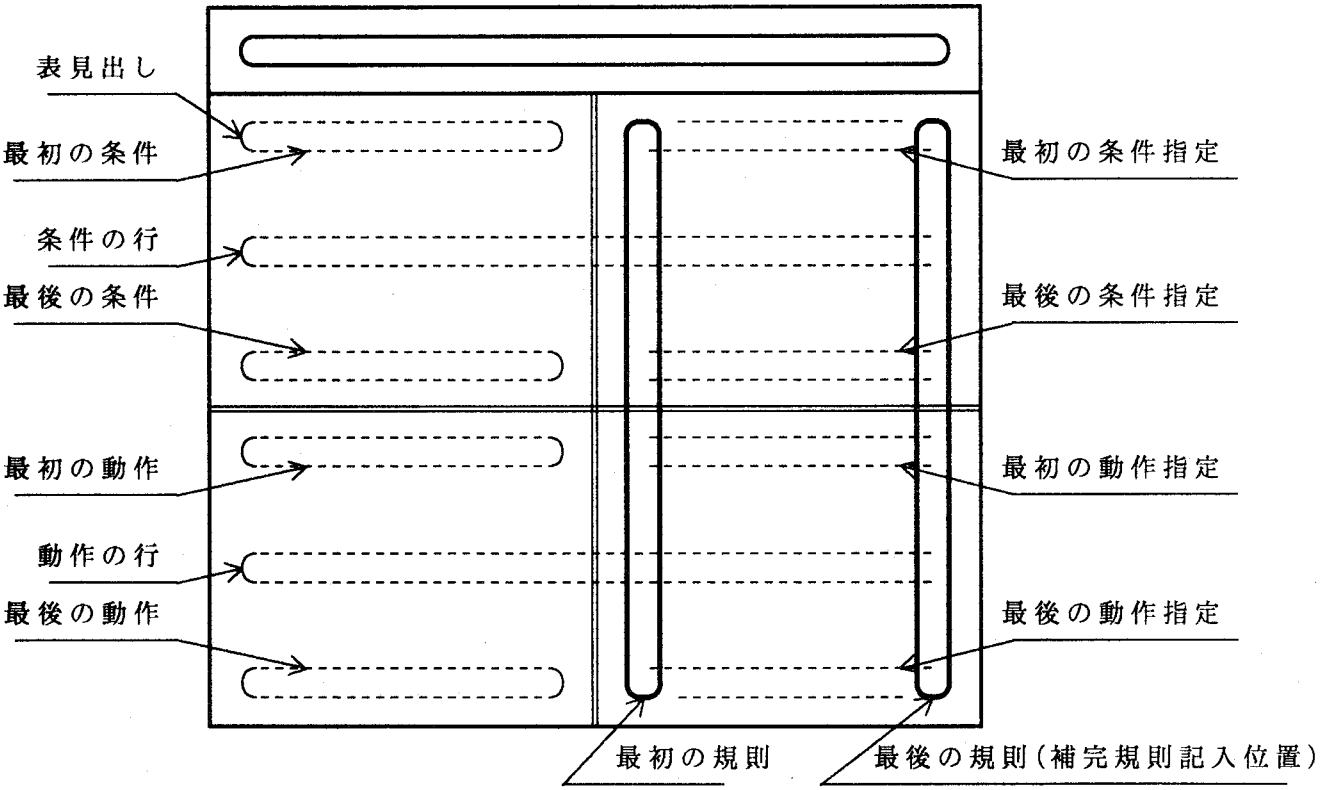
(12)**表見出し (table heading)** 他の文章から決定表を参照するための名前またはその他の手段。それの代わりに、またはそれに添えて、表の明確な説明を書いても良い。

(13)**初期化部 (initialization section)** 最初の条件を調べる前に順番に実行すべき無条件動作を列挙したもの。これは、表見出しの次の行に書くことができる。初期化部は、任意選択とする。

(14)**制限指定表 (limited entry table)** 全ての条件と動作が、指定部の規則への参照なしに、記述部中の記述だけで完成している決定表。

(15)**拡張指定表 (extended entry table)** 記述部中では条件や動作の共通部分だけが、記述されている完成していない決定表。その記述は、指定部の規則で指定された値によって完成する。

図　一般形式と名称　　その２



② 条件指定の書き方

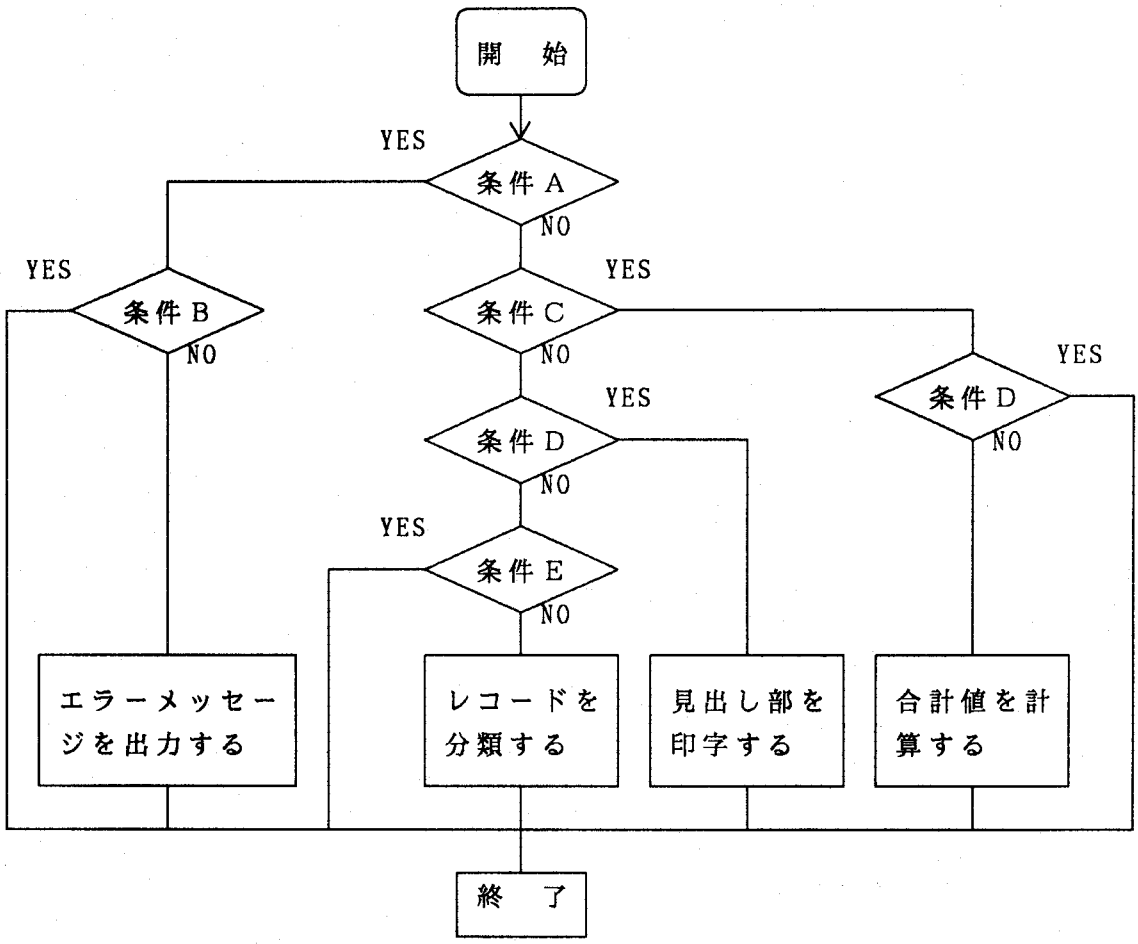
指定形式	規則における意味	適用対象
Y	その規則を満足するには、このYに対応する条件が満たされなければならない（Y＝“Yes”）	制限指定
N	その規則を満足するには、このNに対する条件が満たされてはならない（N＝“No”）	
語句、値又はコード	記述が未完成の条件を、この語句、値又はコードによって完成させる。その規則を満足するには、これにより完成された条件が満たされなければならない。 コードを用いる場合には、その意味を注記する。	拡張指定
－	その規則を満足するには、この“－”に対応する条件は無関係であるか、又は、その規則の文脈においては、この条件は論理的に起こり得ない。起こり得ないことを強調する目的で、“－”の代わりに“＃”を書いてもよい。	あらゆる種類

③ 動作指定の書き方

指定形式	規則における意味	適用対象
X*	その規則を満足されると、記述された動作が実行される。	制限指定
語句、値又はコード	記述が未完の動作を、この語句、値又はコードによって完成させる。その規則が満足されると、この完成された動作が実行される。 コードを用いる場合には、その意味を注記する。	拡張指定
－	その規則が満足されると、記述された動作は実行された動作は実行されない。	あらゆる種類

④　決定表利用イメージ

条件Ａを満足する	Ｙ Ｙ Ｙ Ｙ Ｙ Ｙ Ｙ Ｙ Ｎ Ｎ Ｎ Ｎ Ｎ Ｎ Ｎ Ｎ Ｎ Ｙ Ｎ
条件Ｂを満足する	Ｎ Ｎ Ｎ Ｎ Ｎ Ｎ Ｎ Ｎ Ｙ Ｎ Ｙ Ｙ Ｎ Ｎ Ｙ Ｙ Ｎ Ｎ Ｙ －
条件Ｃを満足する	Ｙ Ｙ Ｙ Ｙ Ｎ Ｎ Ｎ Ｎ Ｎ Ｎ Ｎ Ｎ Ｎ Ｙ Ｙ Ｙ Ｙ － Ｙ
条件Ｄを満足する	Ｙ Ｙ Ｎ Ｎ Ｎ Ｎ Ｙ Ｙ Ｎ Ｎ Ｙ Ｙ Ｙ Ｙ Ｎ Ｎ Ｎ Ｎ － Ｙ
条件Ｅを満足する	Ｙ Ｎ Ｙ Ｎ Ｎ Ｙ Ｎ Ｙ Ｎ Ｎ Ｙ Ｎ Ｎ Ｙ Ｙ Ｎ Ｙ Ｎ － －
エラーメッセージを出力する	Ｘ Ｘ Ｘ Ｘ Ｘ Ｘ Ｘ Ｘ － － － － － － － － － － －
レコードを分類する	－ － － － － － － － Ｘ Ｘ － － － － － － － － －
見出し部を印字する	－ － － － － － － － － － Ｘ Ｘ Ｘ Ｘ － － － － －
合計値を計算する	－ － － － － － － － － － － － － － Ｘ Ｘ Ｘ Ｘ － －
処理を終了する	－ － － － － － － － － － － － － － － － － － Ｘ Ｘ



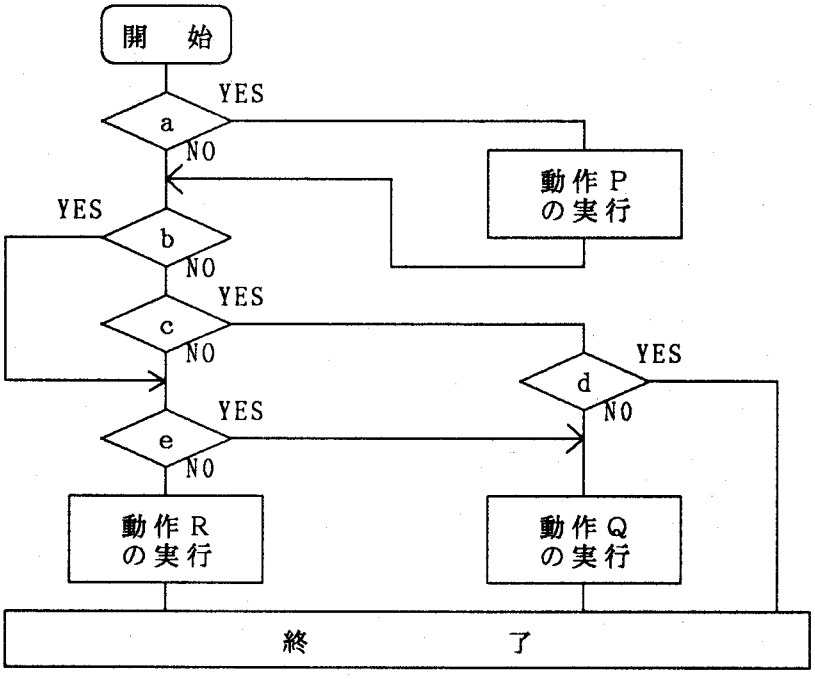
ソフトウェアの仕様を伝えるときには、文章や、絵、フローにも限界があるため、時と場合に応じて色々な表現法を用いることになる。決定表もそのような手段の一つであり、アルゴリズムの説明やプログラムの設計の際に、主体となっている仕様記述の補助として用いることができる。

例えば、処理フローには機能の大まかな流れを記しておき、処理方式の詳細について決定表を用いるとプログラムを作るときにも、テストケースを作るときにも大きな効果がある。

≪小問題≫　（９１年度　第１回　２種試験　午前）

決　定　表

条件１	N	N	N	N	Y	Y	Y	Y
条件２	N	N	Y	Y	N	N	Y	Y
条件３	N	Y	N	Y	N	Y	N	Y
動作Ｐ	－	X	－	X	－	X	－	X
動作Ｑ	－	X	X	－	－	X	－	X
動作Ｒ	X	－	－	－	X	－	X	－



4．コード設計、帳票設計

コード設計、帳票設計に関しては、第２種情報処理技術者試験の出題範囲となっている。しかし、この説明にいきなり入ると混乱が生じる。まずシステム設計はどのような手順で行うべきかについて概要を知る必要があり、その全体の中で上記両設計をどうすべきかを考える必要がある。

←――――――― システム開発過程 ―――――――→

基本計画	概要設計	詳細設計	プログラム開発	システムテスト	導入と運用
システム分析 要求定義	①サブシステム設計 ②出力設計 －ｱｳﾄﾌﾟｯﾄ帳票 －帳票出力装置 －出力ﾚｲｱｳﾄ ③入力設計 －伝票収集分析 －入力媒体 －入力原票設計 －画面設計 ④コード設計 －コードの目的 －コードの機能 －コードの種類 －コード化の手順 ⑤ファイル設計 －マスタファイル －ﾄﾗﾝｻﾞｸｼｮﾝﾌｧｲﾙ －その他ファイル －ファイル編成 －ファイル仕様設計 ⑥ジョブ設計	①帳票レイアウト作成 －出力項目の編集形式 －印刷の開始位置 －改ページ条件 －出力桁数、行数 ②画面レイアウト作成 －出力項目の編集形式 －ファンクションキー －エラーメッセージ ③入力詳細設計 －入力原票 －入力データチェック ④ファイル詳細設計 ⑤プログラム構造設計 －プログラム分割 －モジュール分割			

（１）コード設計

コードとは、対象を表現するために系統化し、効果的に処理（入力処理の簡便化やコンピュータ処理の効率化）できるよう構成する特定の符号のことである。

コードは大別すると、識別機能、分類機能、配列機能からなる。

① 識別機能　他のデータと区別することである。顧客コード「１００１」の大洋工業㈱と「２００１」の大洋工業㈱は社名でなくコードで識別する。

② 分類機能　対象をグループ化することである。例えば、業種別分類では、鉄鋼業を「１×××」に、石油精製業を「２×××」と付番する。

③ 配列機能　データの並び順を表す。例えば、全国の支店番号を北から若いコードを付ける。

コードの付け方には以下のようなケースが考えられる。

● シーケンスコード（一連番号式）

対象データに一定の順番で、先頭から一連の番号を付ける。例えば、発生順、日本語の五十音順など。

（例）社員コード　○○（入社年度）　○○（専攻）　○（性別）　○○（連番）

得意先コード　○○○○（コード）　○○○○（得意先名）

● グループクラシフィケーションコード（十進組別分類式）

分類項目が多い場合、大分類、中分類、小分類のような区分を行う。各桁には意味を持たせる。

（例）組織コード

第１分類		第２分類		第３分類	
本社	1	人事部	1	人事課	1
		総務部	2	庶務課	1
					2
				勤労課	3
		企画部	3	企画課	1
					2

● ニーモニックコード（記憶標識）

文字、数字を組み合わせ、対象に関係ある名称や略号をコードに組み入れたもの。それにより対象を思い起こし易くしている。

（例）家電製品コード

ビデオノーマル型	ＶＤＮ００１
ビデオハイファイ型	ＶＤＨ００１－Ｆ
ビデオハンディ型	ＶＤＨ００１

● ブロックコード（組別式／区分式）

一連の番号コードで複数の分類項目を表せる。少ない桁数で分類項目が表せ、組別（ブロック）毎の追加が容易である。

総務部	０１	人事課	人事課	１１	人材課
	０２	経理課		１２	給与課
	０３	管理課		１３	厚生課
	：	：		：	：

コード化は以下のように進める。

(a) コード化の対象の決定

㋐製造元、㋑商品分類、㋒商品の大きさ、㋓商品の重さ

(b) コード化の範囲（コード化されるデー多数）の決定

㋐製造元　［２００社］、　㋑商品分類　［５０分類］

㋒商品の大きさ　［１～３桁］、　㋓商品の重さ［２桁］

(c) コードの種類の決定

ＯＯＯ－ＯＯ－ＯＯＯ－ＯＯ

| ↓ 重さ（数字またはアルファベット）

| ↓ 大きさ（数字またはアルファベット）

↓ 商品分類（アルファベット２文字）

製造元（アルファベット２文字）

ＡＢＣ－ＴＶ－０２１－３０

ＸＹＺ－ＰＣ－６８０－２５

（２）帳票設計

システム開発における概要設計段階では、利用部門に対してシステムが情報をどのような形式で提供するかを決める。このフェーズでは、利用者がどのような情報を、どのような媒体にほしいかを確認してから、用紙の種類やプリンタが選定され、また、出力仕様として大まかな出力レイアウト（出力イメージ）が作成される。なお、画面に対する出力などもこれに準じて行われる。

詳細設計段階になると、上記出力イメージをもとに、帳票設計や画面設計が行われる。ここでの帳票設計では、スペーシングチャート（spacing chart）上に詳細にレイアウトを記入する。この様式はコンピュータにより出力処理ができる形になっている。すなわち、どのような出力項目をどこに配置するか、用紙の形式をどうするかについても定める。

帳票設計時に明らかにする項目は以下の通りである。

① 出力項目の編集形式（数字、英字、日本語文字、図表など）
② 印刷の開始位置、頁変えのタイミング
③ 見出し（表題、各項目など）
④ 出力項目の配置や桁数
⑤ 印字行間隔、１行の表示桁数
⑥ １頁の印字行数
⑦ レコードの出力順序
⑧ 合計の出し方

用　語

ターンアラウンドタイム、スループット、トランザクション、
チューニング、ベンチマークテスト、
ＲＡＳＩＳ、信頼性、可用性、保守性、完全性、機密性、
ＭＴＢＦ、平均故障間隔、ＭＴＴＲ、平均修復時間、稼働率、
直列システム、並列システム、
決定表、デシジョンテーブル、
コード設計、帳票設計、一連番号式、シーケンスコード、十進組別分類式、
グループクラシフィケーションコード、記憶標識、ニーモニックコード、
ブロックコード、スペーシングチャート

第2種情報処理技術者試験

（１）ソフトウェア
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。

（２）情報処理システム
③　システムの評価に関すること。
　　応答時間、アクセス時間など。

この他、決定表、ＲＡＳＩＳに関する問題はよく出題される。

第９章　情報処理関連英文の読み方

指導目標

コンピュータ実務作業者は英文のマニュアルを読む機会も少なくないが、コンピュータ・カレッジにおいてはそのような場面には出会いにくい。その意味で、授業の中で如何に英文に関する情報を織りまぜていくかが、コンピュータ英文に慣れる鍵ともなる。

情報処理の世界では、ＪＩＳにおける日本語としての用語の規定もあるが、慣用的には英語そのままで用語として用いる場合が多い。したがって、できるだけ新しい用語が出てきたら日本語と元の英単語との対応関係をはっきりととっておくことが重要である。

本章では、２種情報処理技術者試験における英文問題に関する対処の仕方を中心に記述するが、コンピュータ英文に関しての指導は、この章を通してだけではなく、むしろ全章を通じてできる限り元の語句や字句について両語間での対応をとっておくべきことが肝要である。

英文になれるポイントとしては、

○　用語について、新たに登場するたびに、両語間で正しい対応関係をとること

○　コンピュータ関係の基礎知識に関して、簡単な英文が素早く読めるようにしておくこと（但し、日本語でのコンピュータの基礎知識を十分蓄積しておくことが前提）

である。

指導内容

1．２種試験における英文問題について

午前の部の後半（第１１問～２０問）の１０題中５題が任意選択となっており、最後の２問（第１９、２０問）は必ず、英文に関する選択問題である。これらは、日本語で出題されれば、比較的やさしい問題に属するはずで、是非２題とも選択し、これらについては正解を得てほしいところである。

2．試験問題のパターン

①　２～３行での説明的な英文を読み、対応する語句や字句を解答群から選択させる

②　１０数行の英文を読み、大意を表す日本語訳を解答群から２～３個選択させる

③　２～３行の説明的な英文中に、穴があり、それに適する単語を解答群から選択させる

④　１０数行の英文に所々穴があり、それに適する語句、字句を解答群から選択させる

時間配分から、１問１０分以内には解かなくてはならないであろう。例年、①または③のパターンと、②または④のパターンが１問ずつ出題される傾向にある。①、③のパターンでは中がさらに５～６個の文章から構成される。

3．英文のレベル

○　英文に関して、文法的には高等学校２年程度の表現である

○　英単語に関しては、高等学校３年程度もしくはそれ以上のものもときたま含まれる

○　学校英語とは全く別の次元で、コンピュータ用英単語が当然のように出てくる

4．問題の解き方

パターン① ２～３行での説明的な英文に対応する語句や字句を解答群から選択

(1) 解答群の語句、字句をさっと見て、グループ分けをすると共に、どんなことに関する説明文であるかを感じとる（読み取る）

(2) 各説明英文の中のポイントとなる単語を探す

(3) グループ化された解答と対応付けを行う

（平成４年度第１回午前問１９）の例

問１９　次の英文ａ～ｅの意味にそれぞれ最も近い字句を、解答群の中から選べ。

a　A **computer** that combines the elements of both digital and analog techniques.

b　A **language** for programs that can be expressed directly in a binary format acceptable to the central processing unit.

c　A **method** of designing a program,and associated data structures,that improves clarity,reduces complexity and renders it more amenable* to modification and debugging.

d　The **processing** of data where a number of similar input items are grouped for processing during the same machine run.

e　Special purpose **hardware or software** that enables one system to act as if it were another. It is used,for example,to minimize reprogramming effort when a new computer replaces an existing one.

* amenable 扱いやすい

解答群

ア　assembly language	イ　batch processing
ウ　concurrent programming	エ　emulator
オ　host computer	カ　hybrid computer
キ　interpreter	ク　machine language
ケ　structured editor	コ　structured programming

（注）問題文中には、網掛けとアンダーラインはない。

解法手順

(1) 解答のグループ分け

（この分類に際しては下記単語について知識があることが前提となる）

① ア assembly language 、　　ク machine language

② イ batch processing

③ ウ concurrent programming、　　コ structured programming

④ エ emulator

⑤ オ host computer、　　カ hybrid computer

⑥ キ interpreter

⑦ ケ structured editor

(2) ポイントとなる単語や文節を探す

前頁問題文中の、網掛け部分、１点鎖線アンダーライン

(3) グループ化された解答と対応付けを行う

a： まず、⑤のグループに対応するとみる。ディジタルとアナログの両技術という点から、ハイブリッド型のコンピュータを思い起こすことができる。

b： まず、①のグループに対応するとみる。２進形式表現という点から、機械語になるはずである。

c： 最初の１行で何らかの方法ということで、③のグループが対応しそうである。分かりやすく、複雑さを減らし、また、変更やデバッグを容易にするという点から、構造化プログラミングが選択される。

d： まず、“processing”という単語からバッチ（一括）処理が対応しそうなことはすぐ読めるが、同じマシン上での処理という部分から、concurrent programmingと早合点しないように。

e： 特別なハードウェアあるいはソフトウェアという点から、emulator、interpreter、structured editorのいずれかとなろう。エミュレータについてはっきりと知らなくても、他のシステムと同じ働きをする部分から、structured editorは外され、また、インタープリータはソフトウェアでしか有り得ないことを知っていれば解ける。

パターン② １０数行の英文の大意を表す日本語訳を解答群から選択させる

(1) 解答群の大意をさっと見て、どんなことについての話であるかを推測する。勿論キーとなる単語はある程度知っているはずであるから、後はキーワードについての一般知識を活用すればよい。

---

(2) 解答群の各説明に相当する英文の部分との対応付けをする
(3) 日本語訳がそれと、あるいは前後の文脈から正しいかどうか判断する

（平成４年度第１回午前問２０）の例

問２０　次の英文の内容に近いものを、解答群の中から二つ選べ。

1　　The CD-ROM version of The General Software Catalog contains details
2　of more than 10,000 software sold in the U.S.,and demonstration version
3　of major software packages. The disk also incorporates a text-retrieval
4　package,which lets you search the equivalent of four volume books of 500
5　-page each in less than 2 seconds.　At the touch of a button,
6　you can move from relevant product descriptions to running a
7　demonstration of that product.
8　　You can undertake single or complex word searches by simply entering
9　the word or phrase. The package instantly displays the retrieved
A　thesaurus,which helps you look for products or services related to a
B　particular manufacturer's computers.
C　　The retail price is $350,and 50% discount is available for
D　educational institutions.

解答群

ア　このCD-ROMを利用するためには、別売りの検索プログラムが必要である
イ　500ﾍﾟｰｼﾞの本にして４冊分の情報を２秒以内に検索できるプログラムがついている。
ウ　索引がついているので、特定の機種ごとの製品やサービスの一覧も表示できる。
エ　ある製品の説明からボタン一つでそのデモに進むことができる。
オ　価格は350ﾄﾞﾙで、購入者は次から改定版を５割引で入手できる。

（注）問題文中には、網掛けとアンダーライン、および、行番号はない。

＜アンダーライン部分の文の意味＞

① １～３行　　ＣＤ－ＲＯＭ版には、米国で売られている１万本以上のソフトウェアの詳細と主要なソフトウェアパッケージのデモ版が入っている。

② ３～５行　　ディスクにはテキスト検索用パッケージも含まれており、それにより、500ﾍﾟｰｼﾞの本４巻分を２秒以内で探すことができる。

---

③　５～７行　　ボタンに触れると、関連製品の説明からその製品のデモに進むことができる。

④　８～Ａ行　　単純に単語や字句を入れるだけで、一つまたは複数個の言葉を探すことができる。このパッケージは、検索された情報を直ちに表示する。

⑤　Ｃ～Ｄ行　　小売り価格は350ﾄﾞﾙで、教育機関には５０％のディスカウントが適用される。

解答群についての検討

ア：　ＣＤ－ＲＯＭは検索プログラムにより利用の範囲は拡大されるが、問題の英文中では、別売りという表現は出てこない。

イ：　英文中の３～５行目にこれに相当する文章が現れる。

ウ：　添付のパッケージにはシソーラスが組み込まれており、特定メーカのコンピュータに関連する製品やサービスを探す手助けはしてくれるが、一覧の表示までできるとは書かれていない。

エ：　英文中の５～７行目にこれに相当する文章が現れる。

オ：　このような内容を持つ文節は出てこない。

パターン③　２～３行の説明的な英文に関する、穴埋め単語を解答群から選択させる

(1) 問題の英文中にある“穴”が、主語であるのか、目的語であるのか

(2) “穴”が１つの単語に相当する場合、名詞、形容詞、動詞、その他何であるか

(3) 英文の作りとして適切か

等について検討する必要がある。

（平成３年度第２回午前問１９）の例

問１９　次の英文中の　　に入れるべき適切な字句を、解答群の中から選べ。

(1) A　a　computer is one that directly counts the numbers that represent numerals,letters,or other special symbols.

(2) A　b　is a detailed step-by-step set of instructions that cause a computer to function in a desired way.

(3) A rewritten program that's sold to perform a common task is called an applications [c].

(4) The three types of [d] used to classify data items are numeric, alphabetic, and alphanumeric.

(5) The organized collection of software that controls the overall operation of a computer is the [e] system.

(6) Desirable features of analog and digital machines are sometimes combined to create a [f] computing system.

解答群

ア	codes	イ	ＣＰＵ	ウ	device	エ	digital
オ	hybrid	カ	mainframe	キ	online	ク	operating
ケ	package	コ	program				

（注）問題文中には、網掛けとアンダーラインはない。

解答の方法

(1) ～ computer の候補となりうる解答は、[digital、hybrid、mainframe] である。このうち、ハイブリッド・コンピュータの説明にはなっていないのでhybridはまず除外される。数値、文字、その他特殊記号を表す数を扱うのは、メインフレームだけでなく他の種のコンピュータも同様であり、答としては、digitalが正しい。

(2) コンピュータに対して希望する処理を行うステップバイステップ命令セットとあれば、[codes、package、program] のいずれかとなろう。文法的には単数形名詞と考えるとcodesは除外される。packageでもおかしくはないが、より一般的な説明としてはprogramが最も適切である。

(3) アプリケーションプログラム、アプリケーションパッケージ、アプリケーションコード等はほとんど同じものを意味しているが、市販される点で、アプリケーションプログラムが適切な言葉である。

(4) データ項目は、数字、英字、英数字に分類できるが、これを一般にコードという。

(5) コンピュータ上でのあらゆる操作をコントロールするソフトウェアは、オペレーティングシステム以外にない。

(6) アナログとディジタルの両機能を持つシステムをハイブリッドという。

上記(4)～(6)はコンピュータに関連する一般的な基礎知識を持っていることが問題を解ける条件である。

5．ふだんからの準備

コンピュータ処理のための英語力を強化する上で、ハードウェア、ソフトウェアに関する基礎知識を学習する過程で、以下の点に注力させる必要がある。

① まず、コンピュータの基礎的な知識について、正しい知識を入れること

② コンピュータ用語については、日本語、英語の対応関係を必ず付けておくこと

② 英単語の場合、省略形、フルスペル両方覚えておくことが重要である

③ どのような英文に慣れておくべきかは、２種の受験雑誌等を参考とするのがよい

④ コンピュータ辞典で、用語やコンピュータに関連する話について知識を入れておくこと（これは、他の問題を解く上でも重要である）

⑤ 参考書として、例えば、共立出版株式会社発行の、下記図書が推奨される。

- 対話・解説コンピュータ英語
- コンピュータ英語ハンドブック
- コンピュータ用語の基礎知識
- コンピュータ英和・和英辞典

付録　第２種情報処理技術者試験の出題範囲に対応する事項

＜ソフトウェアの基礎知識＞に関連する事項

	出　題　対　象	関連章節
ソフトウェア	①　オペレーティングシステム（制御プログラム）に関すること。 (1) 目的、機能、役割 → 第１編　第６章　第１～４節 (2) 歴史 → 一般参考図書等を参照されたい	
	②　プログラム言語に関すること。 アセンブラ、コンパイラ、インタプリータ、ジェネレータなどの言語の特徴、歴史、用語 → 第１編　第６章　第５節	
	③　ファイル編成に関すること。 (1) 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 → 第１編　第３章　第６節 → 第２編　第３章　第１～10節 (2) 各装置の容量計算、データ転送 → ハードウェアの基礎編を参照されたい	
	④　ユーティリティに関すること。 各種ユーティリティプログラムの使用目的、使用方法、用語など。 → 第１編　第６章　第６、７節	
	⑤　アプリケーションに関すること。 各種アプリケーションプログラムの使用目的、使用方法、用語など。 → 第１編　第５章　第１～３節	
	⑥　アルゴリズム（算法）に関すること。 表引き（テーブルサーチ）、突合せ、整列（分類）など一般によく使用されるアルゴリズム及びその図式化など。 → 第２編　第１章　第１～３節	
	⑦　データ構成、データ様式に関すること。 (1) ファイルの項目、レコード、ブロック → 第２編　第３章　第１～２節 (2) コード設計、帳票設計 → 第１編　第８章　第４節	
	⑧　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など。 → 第１編　第４章　第２～３節	

ソフトウェア	⑨ プログラム技法に関すること。 プログラムを作るための効果的な手法など。	→	第２編　第２章　第３節
	⑩ 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１編　第４章　第１、４節
	⑪ 伝送制御に関すること。 (1) 伝送制御手順（ＨＤＬＣ手順、基本系データ伝送制御手順など）	→	第１編　第７章　第５節
	(2) プロトコル	→	第１編　第７章　第５節
	(3) 伝送速度、誤り制御	→	第１編　第７章　第４節
	⑫ データベース	→	第２編　第４章　第１～３節
情報処理システム	① システム処理形態に関すること。 オンラインシステム、バッチ処理など。	→	第１編　第７章　第１節
	② システム構成に関すること。 デュプレックスシステム、マルチプロセッシングなど。	→	第１編　第７章　第２節
	③ システムの性能評価に関すること。 応答時間、アクセス時間など。	→	第１編　第８章　第１節
	④ その他 (1) オフィスオートメーションシステム	→	第１編　第７章　第１節
	(2) ファクトリーオートメーションシステム	→	第１編　第７章　第１節
	① 情報処理関連英文の読み方		第１編　第９章