高校普通教科「情報」新・試作教科書

2006.12.11 版
(2007.1.17 訂正)

情報処理学会初等中等教育委員会

はじめに

本「新・試作教科書」は、近い将来において高等学校普通教科「情報」の内容がどのようなものであるべきかについて、情報処理学会初等中等教育委員会試作教科書ワーキンググループにおいて検討した成果の一環として取りまとめたものである。我々としては「情報 I」「情報 II」をそれぞれ次のように位置づけている。

- 「情報 I」 — 全員必修の 2 単位科目であり、2003 年指導要領の「情報 B」「情報 C」の内容をおおむね含んでいる (高度な部分、分量上含めるのが難しい部分を除外している)。
- 「情報 II」 — 2 単位の選択科目であるが、高等教育に進む生徒全員が履修することを強く希望する。2003 年指導要領の「情報 B」「情報 C」の内容をさらに深めたものとなっている。

我々はこのほかに「情報 IIIx」についても構成案を検討したが、テキストの執筆には至らなかった。これらについての詳細については、「教科『情報』新・試作教科書の提案」(情報処理学会 情報処理教育委員会「高校教科『情報』シンポジウム 2006」資料集 pp.60-79) において解説しているので、そちらを参照されたい。

本「新・試作教科書」は、個人での複製・利用については自由におこなってよいものとする。また、学校・企業その他において、教育を目的として複製・利用する場合は、(1) 配布対象・教育の内容・複製部数を連絡していただき、(2) 配布時に由来を説明して頂くことを条件に、自由な利用を認める。また必須の条件ではないが、(3) 利用結果について概要を報告頂けることが望ましい。

(試作教科書 WG メンバー)
久野　靖 (筑波大学)
兼宗　進 (一橋大学)
西田　知博 (大阪学院大学)
山之上　卓 (鹿児島大学)
小井土　正範 (長野県上田千曲高校)
神沼　靖子 (情報処理学会フェロー)
辰己　丈夫 (東京農工大学)
奥村　晴彦 (三重大学)
長　慎也 (一橋大学)
中野　由章 (千里金蘭大学)
並木　美太郎 (東京農工大学)
和田　勉 (高麗大学/長野大学)
(連絡先メールアドレス)
joho-text@ipsj.or.jp

目 次

第Ⅰ部

情報Ⅰ

第1章　ネットワークと情報

1.1　コンピュータネットワークの利用

1.1.1　ネットワークの基本概念

コンピュータネットワークの定義

ネットワーク (network) とはもともと「網状のもの」という意味であり、そこから網目状に結びついたさまざまな事柄を指し示す言葉として用いられている。友達どうしの情報網もネットワークであるし、多くの線路によって結ばれている鉄道網もネットワークである。

コンピュータネットワーク (computer network、以下では混同の恐れがない場合は単に「ネットワーク」と記す) とは、コンピュータなどの情報機器を互いに接続することによってできる通信網のことであり、接続された機器どうしの通信ができることで、情報機器の利便性や可能性が大きく広がった。以下ではネットワークのさまざまな面について学んでいく。

ネットワークの構造

家庭、学校、会社内などの比較的狭い範囲で作られるネットワークを **LAN**(Local Area Network) と呼ぶ。これに対し、広い範囲にまたがるネットワークを **WAN**(Wide Area Network) と呼ぶ (図 1.1)。

複数のネットワークを相互に接続すると、それらのどこに接続された機器どうしでも通信ができ、全体として 1 つの大きなネットワークとして働く。たとえば、大規模な LAN は通常、小さな LAN の集まりでできているし、WAN も複数の LAN を相互接続して作られていることが多い。

さらに WAN どうしを接続してより大きなネットワークを作ることもできる。このようにして、世界全体にまたがるネットワークを形成したものが**インターネット** (The Internet) である。個人の家庭や多くの組織は**プロバイダ** (インターネット接続業者) が構成する WAN を経由してインターネットに参加している。

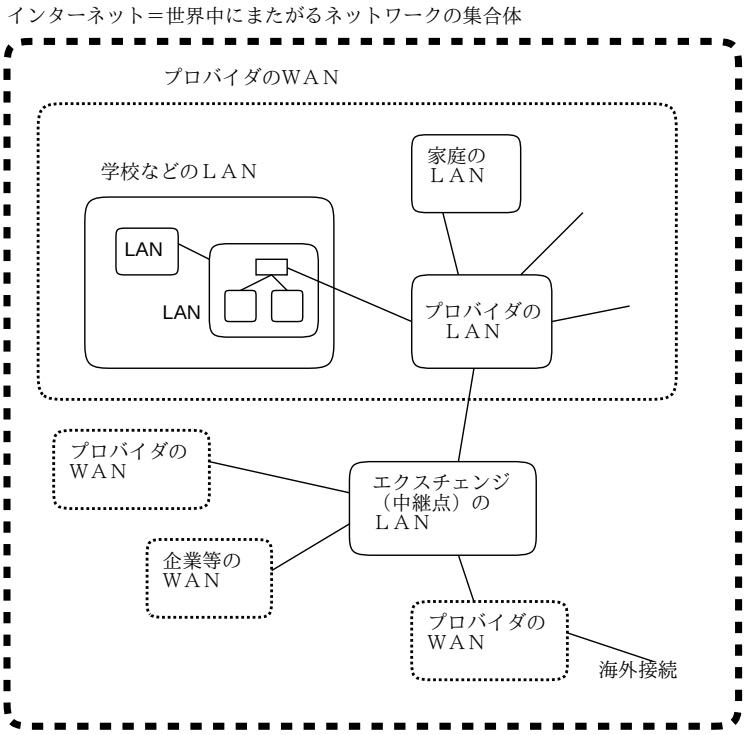


図 1.1: LAN、WAN、インターネット

ネットワークのアドレス：ドメイン名

我々は葉書や手紙を送るときに、相手の住所と名前を書くことによって送り先を指定する。それと同じように、ネットワーク上で通信する場合も、相手先を特定する住所に相当する**アドレス**が必要になる。インターネットに接続されているコンピュータのアドレスは**ドメイン名**と呼ばれるもので指定できる。

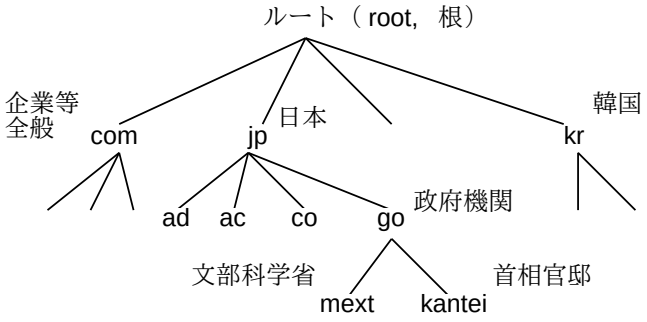


図 1.2: ドメインの階層構造

ドメイン名は、いくつかの名前 (ラベル) が、「.」によって区切られた形で表現される。それぞれのラベルは、右から左に向かって次第に大分類から小分類を表すようになっている。たとえば、「kantei.go.jp」というドメイン名は、左から順に国名 (日本) を表す「jp」、組織の種類 (政府機関) を表す「go」、組織の名前 (首相官邸) を表す「kantei」の 3 つのラベルによって構成されている。これらのラベルによる分類は、図 1.2 に示すような階層構造 (木構造) をなしている。

最も左のラベルは、階層構造の最上位となるので、**トップレベルドメイン** (TLD) と呼ばれ、そのドメインが属する国などを表す最も大きな分類となる。2 文字の TLD は国を表し、jp が日本 (Japan)、uk が英国 (United Kingom)、kr が韓国 (Korea) などとなっている。

(コラム: ドメイン名とその管理)
米国は本来 us(United States) であるが、インターネットの発祥の地が米国である関係から、米国では edu が教育機関、gov が政府組織など分野ごとに特別の TLD を使う。com(商用組織)、org(非営利団体) なども同様であったが、現在では米国以外の組織でも取得でき、**汎用トップレベルドメイン** (gTLD) と呼ばれる。
ドメインをどのように細分化するかは、各 TLD を管理する組織に委ねられる。組織や個人は管理組織に依頼してドメイン名を取得し、利用することができる。

ネットワークのアドレス:IP アドレス

ドメイン名は人間にはわかりやすい形で表現されているが、コンピュータで効率よく処理するには向いていない。そこで、コンピュータどうしのやり取りでは、決まった短い長さ、具体的には 4 バイト (1 バイトは 8 ビット) の **IP アドレス**によって通信相手を特定する。IP アドレスは通常、各バイトの値を 0〜255 の数値であらわし、それらを「.」でつなげてたとえば次のような形で表記する: [1]

```
192.0.34.166
```

ドメイン名と IP アドレスの対応は **DNS**(Domain Name Service) と呼ばれるしくみを使って調べることができる。これは、ドメイン名が図 1.2 で示すような階層構造になっていることを利用し、ドメイン名と IP アドレスの対応を保持した一種の分散データベースを参照することにより、特定のサーバに負荷を与えずに世界中から任意のドメイン名が検索できるしくみとなっている。

プロトコルとサービス

ネットワークでつながれたコンピュータや情報機器は、その情報を決められた約束に従ってやり取りする。この約束のことを**プロトコル** (protcol) と呼ぶ。インターネットに接続されている機器は、**TCP/IP** と呼ばれるプロトコルを用いて通信を行う。

TCP/IP では、すべての通信は**パケット**と呼ばれる一定サイズ以下の「かたまり」単位で行われている。これを**パケット交換**と呼ぶ (図 1.3)。パケットに入らない大きなデータは、送信側でパケットに分解して送り、受信側で元の形に組み立てる。パケット交換方式では、複数の箇所からのデータが途中の経路を「相乗り」でき、通信路を効率的に利用できる (ただし混雑してくると全体に通信が遅くなる)。[2]

インターネットは世界中の任意のコンピュータどうしが TCP/IP を用いて通信できるようにしてくれているが、実際に電子メール、WWW、

[1] IP バージョン 4 の場合。より長いアドレスを使う IP バージョン 6 も使われ始めている。

[2] 電話などは最初の呼び出し時に 1 本の線を割り当て、通話中はその線を占有する。これを**回線交換**と呼ぶ。回線交換では混雑してくると「話し中」になりつながらなくなるが、すでにつながっている通話は (専用の線が割り当てられているので) 遅くなることがない。

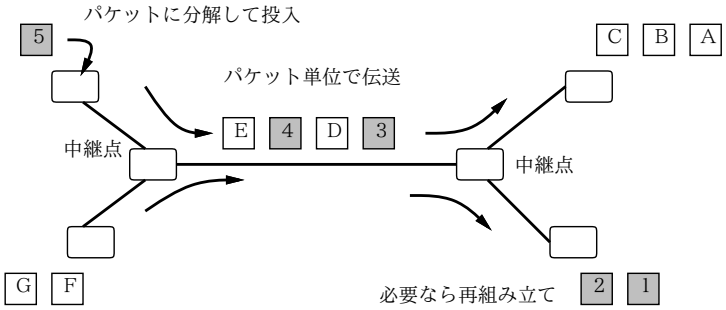


図 1.3: パケット交換

ネット電話など、さまざまな**ネットワークサービス**を利用できるためには、これらのサービスを提供してくれる**サーバ**と呼ばれるコンピュータも必要である。ユーザが手元のコンピュータ上でサービスを利用するためのソフトウェアを動かすと、これらのソフトウェアがネットワーク経由でサーバ上のプログラムに接続し、データをやりとりすることでサービスが利用できる。

ネットワークと接続機器

コンピュータなどの情報機器を LAN に接続するためには、**ハブ**と呼ばれる機器を用意し、各機器とハブの間をネットワークケーブル (UTP ケーブル、図 1.4) で接続する。ハブは、そこに接続された任意の機器どうしの間で信号をやりとりする機能を提供している (図 1.5)。

複数のネットワーク (LAN など) を相互に接続するのには、**ルータ**と呼ばれる機器を使用する。ルータは宛先までに複数のネットワークを経由する必要のあるパケットを正しい方向に中継していく働きを持つ。

図 1.4: UTP ケーブルとコネクタ

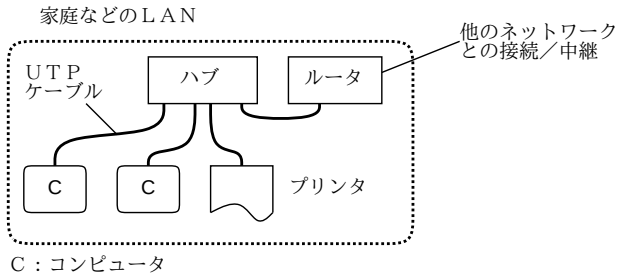


図 1.5: 小規模な LAN とハブ/ルータ

家庭用の小型のルータでは、家庭内の LAN とプロバイダが提供する WAN との間の中継のみを扱うように簡単化されていて、さらにハブを別に買わなくても済むように LAN 側のハブも内蔵されているものが多い。

最近では、LAN の通信にケーブルの代わりに電波を用いる**無線 LAN** が普及してきている。また、駅、空港、飲食店などの公共の場所でも、ホットスポットと呼ばれる公衆利用可能な無線 LAN サービスが提供されている。

(コラム: 電波の干渉)
無線 LAN は電波を使って通信を行うため、同じ周波数帯を使用する機器がある場合は、電波の干渉が起こり、通信速度が落ちたり、通信が出来なくなったりする。無線 LAN の規格のうちには、電子レンジの電波などと周波数が重なっているものもあり、このため、たとえば、電子レンジを使用中には無線 LAN の通信がうまく行えないことがある。

1.1.2　電子メール

電子メールとは

電子メール (electric mail, e-mail) とは、コンピュータで作成した「電子的な手紙」をネットワークを用いて個人から個人へ送信するサービスである。送信するメッセージはテキストが中心だが、画像や音声などの情報も付加することができる。

人手で運搬する必要がある紙の手紙とは違い、電子メールはネットワークに接続されている場所であれば、世界中のどこであっても短時間でそのメッセージを送ることが出来る。また、宛先を複数指定すれば、同じ内容のメッセージを 1 度に複数人にまとめて送ることも可能となる。

電子メールのアドレスは、一般に次のような形をしている:

ユーザ名@ドメイン名

たとえば「jun@example.net」の場合、jun がユーザ名、example.net」がドメイン名である。ここでドメイン名は、メールを受け取る相手が使っているメールサーバを (直接もしくは間接的に) 指定し、ユーザ名はそのメールサーバを使っている「誰」宛かを指定する。

電子メールの配送とヘッダ

電子メールでは、To: (宛先)、Cc: (写し)、Bcc: (秘密の写し) の 3 通りの方法で送り先を指定できる。送り手がメールメッセージを送信すると、それを手元の**メールサーバ**が受け取り、これらの宛先情報を見て、(複数あればそれぞれの) 相手先のメールサーバにメッセージを送ってくれる。メッセージは相手先のメールサーバで保管され、相手は自分のメールサーバからメッセージを取り寄せて読む (図 1.6)。

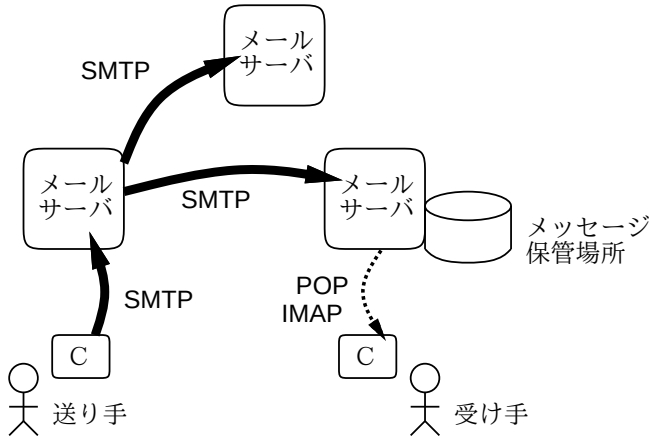


図 1.6: メールサーバによるメッセージの中継

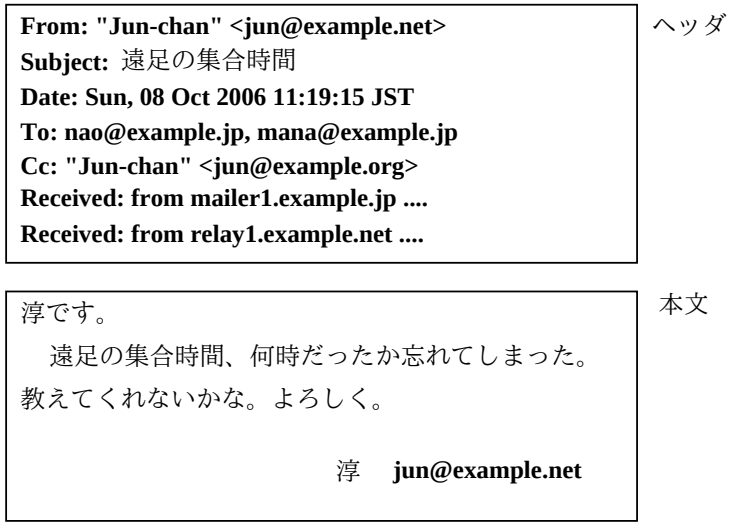


図 1.7: 電子メールメッセージの構成

送り手から手元のメールサーバ、手元のメールサーバから相手のメールサーバへの通信には **SMTP**(Simple Mail Transfer Protocol) と呼ばれるプロトコルが使われる。メールサーバからメッセージを取り寄せるのには **POP**、**IMAP** などのプロトコルが使われる。

電子メールのメッセージは**ヘッダ**とその後に続く**本文**に分かれている (図 1.7)。本文はメッセージの内容に相当する。ヘッダはメールサーバが扱う部分で、メールサーバはメールを中継するときにここにメッセージを扱った日時やどこから受信したかなどの情報をに記録していくので、受信したメッセージのヘッダ情報を参照すると、メッセージがどのように中継されてきたか知ることができる。表 1.1 に、代表的なヘッダ情報を示す。

ヘッダ情報の中で、メッセージ作成時に利用者が指定するのは、通常、題名 (Subject:) と宛先のアドレスである。宛先のうち、To:、Cc:(carbon copy、昔複写にはカーボン紙をはさんで書いたことから来ている) については、メールサーバは指定したものをそのまま残すので、受信者は誰と誰にメッセージが送られたか (そして誰が宛先で誰が写しなのか) を知ることができる。Bcc:(blind carbon copy) の情報は (秘密の写しなので) 削除され、誰が秘密の写しを受け取ったかは分からないようにされる

表 1.1: 電子メールのヘッダ情報

From:	送信者のアドレス
To:	宛先のアドレス
Reply-To:	送信者側から指定した返信時に送って欲しいアドレス
Cc:	コピーとして送る宛先のアドレス
Bcc:	コピーとして送る宛先のアドレス 電子メールが届いた先ではアドレスがヘッダに残らない
Subject:	メールの題名
Date:	送信日時
Received:	ネットワーク上でメールが受け渡しされた記録 受け渡しが行われるごとに追加される

(そのため、Bcc:で受け取ったメッセージに不用意に返信すると自分が秘密の写しを受け取ったことが分かってしまうので、注意が必要である)。

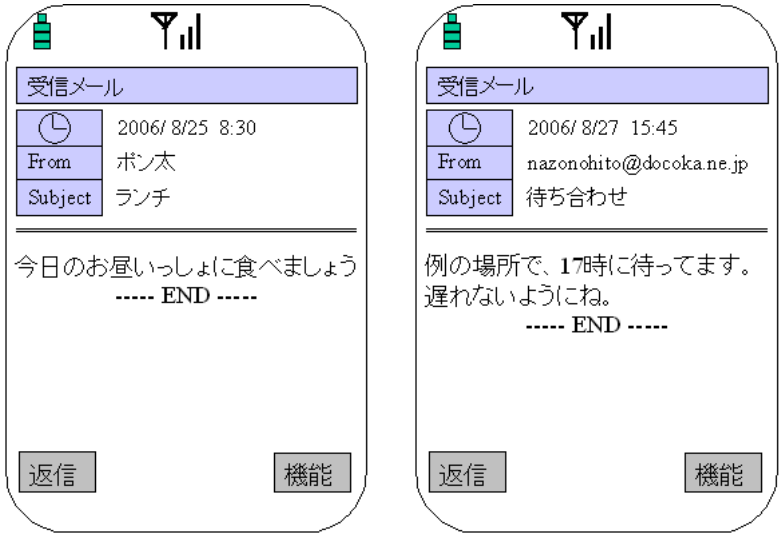
電子メールは文字を基本としたやり取りであるので、メッセージの本文に文字のみを使ったテキストメールが基本である。しかし、**MIME** (Multipurpose Internet Mail Extensions) と呼ばれる形式でデータを文字に変換することによって、画像やワードプロセッサや表計算ソフトで作ったファイルをメッセージに添付して送ることも可能である。

電子メールには利用する機器や、形態によって本文やヘッダ情報などの形式が変わることがある。以下では、その違いについて、具体例を交えて見ていく。

携帯電話を用いた電子メール

携帯電話では、個々の携帯電話会社固有の**ショートメール**などと呼ばれるメッセージ機能もあるが、インターネットの電子メールの方が多く使われている。この場合は宛先やヘッダ等の仕組みは上で説明した一般の場合と同様である。

ただし、携帯電話では保存できるデータ量がそれほど大きくないなどの制約があるため、ヘッダ情報は図 1.8 のように、送信者、題名、送信



(a)　　(b)

図 1.8: 携帯電話へのメール

日時、宛先 (図の例では表示されていない) といった最小限のものしか伝えられない。また、送信者の情報も、アドレスが電話帳に登録されている場合は、メールアドレスの代わりに名前が表示されるなどの機能もある (図 1.8(a))。

この機能は、誰からメールが来たかがわかりやすく、特にメール本文に自分の名前などを書かなくても、コミュニケーションには問題はない。特に、友達や家族などの親しい人とのメール交換では、本文に名前を名乗ることは少なくなると思われる。しかし、これは携帯電話という特性の上に成り立っているコミュニケーション形態であり、パソコン等でやりとりされるその他の電子メール環境では、誰がそのメールを書いているかを名乗るなど、携帯電話上のメール交換とは異なるということを意識しておく必要がある。

パソコンなどを使った電子メール

パソコンなどを用いた電子メールのやり取りはでは、**メーラー** (または電子メールソフト) と呼ばれるソフトウェアを用いてメッセージを読み書きするのが基本的な (古くからある) 方法である。この方法では、ネッ

トワークにつながっていなくてもメッセージを読んだり作成したりできるので (実際にメッセージを送受信するときはネットワークにつながる必要がある)、ノートパソコンなどを利用して移動しながらメールを読み書きする人がとくに多く使っている。

このほか、現在ではブラウザを用いてサーバにアクセスしメッセージの読み書きを行う、**Web メール**と呼ばれる形態で利用することも多い。Web メールでは、サーバにメッセージや個人設定が保存されているので、インターネットに接続でき、Web ブラウザが使える環境であれば、個人環境等の特別な設定をすることなく利用できる。そのため、旅行先等で電子メールをやりとりをするのに便利である。

1.1.3 WWW

WWW の基本概念

WWW(World Wide Web) は、インターネット上の各地で公開されている **Web ページ**と呼ばれる形の情報をネットワーク経由で見ることができるような情報システムである。

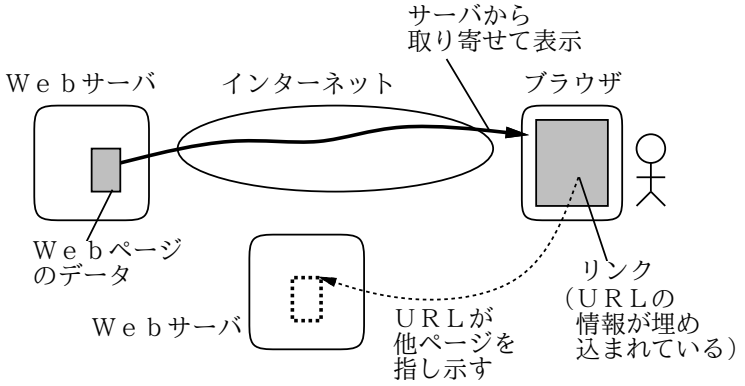


図 1.9: WWW の原理

WWW では、Web ページは世界中に多数稼働している **Web サーバ**によって公開されている。そして、ユーザは自分の手もとのコンピュータ

で**ブラウザ** (Web ブラウザ) と呼ばれるソフトウェアを動かすことで、Web サーバからの情報を取り寄せて読むことができる (図 1.9)。

Web ページの中には他のページ等を指す**リンク**と呼ばれる情報を埋め込むことができ、リンクを通じて多数のページが有機的に結び合わさった構造が作られる (これを**ハイパーテキスト**という)。他のページを指しているリンクを選択すると、そのページがサーバから取り寄せられてきて、これまで見ていたページの代わりに表示される (図 1.10)。

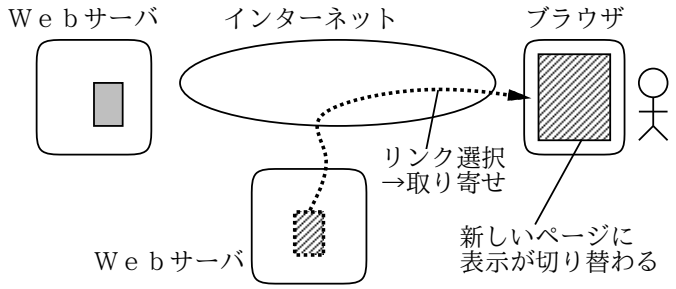


図 1.10: リンクの原理

WWW では、各ページは **HTML**(HyperText Markup Language) と呼ばれる決まった書き方で記述されており、これによりブラウザはどこから取り寄せたページでも読みやすく整形して表示することができる。また、Web サーバとブラウザの間の通信には **HTTP**(HyperText Transfer Protocol) という共通のプロトコルが使われており、これによってブラウザは世界中のどこの Web サーバからでも情報を取り寄せて来ることができる。

また WWW では、リンクなどでさまざまなもの (Web ページ、画像ファイル、サウンドファイルなど、ネット経由で取り寄せられるもの全般) を指定するのに **URL**(Uniform Resource Locator) と呼ばれる決まった形式のアドレスを用いる (図 1.11)。これにより、世界中のどこの Web サーバにあるものでも同じように指定し、必要なら取り寄せることが可能になった。

URL の先頭部分は、どのようにしてその指している対象を取り寄せるかを指定している。ここが「`http:`」となっているものは HTTP プロトコルにより Web サーバから取り寄せる情報である。この場合、「`//`」に続いてその指している対象を取り寄せる Web サーバのドメイン名が

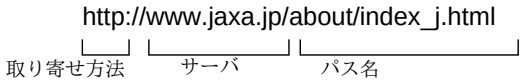


図 1.11: URL の構造

ある。その後「/」以下はそのサーバのどこにある何という名前 (パス名) のものを取り寄せるかを指定している。URL は単にそのサーバに格納されているファイルを指していることもあるし、そのサーバ上で動作し、情報を送り返してくれるようなソフトウェアを指していることもある。

(コラム: WWW のはじまり)
WWW は 1991 年に、CERN(ヨーロッパ粒子物理研究所) に滞在していた研究者 Tim Berners-Lee によって発明された。WWW では、ブラウザ上でリンクを選択していくだけで情報を取り寄せることができる。その結果、インターネット利用者が世界中の情報を手軽に入手できるようになった。さらに、Web サーバに情報を置くことも難しくないため、個人でも簡単に情報発信が可能となった。このため、あっという間に多くの人が WWW を利用するようになり、WWW がインターネットを爆発的に普及させる原動力となった。

WWW の活用

サーチエンジンとは、インターネット上のアクセス可能なページを定期的に訪問し、その情報を検索可能なように整理するプログラムである。サーチエンジンによって集められた情報は、検索サイトによって利用者に提供される (ただし、世界中には膨大な数のページがあるため、その全部を検索できるわけではない)。サーチエンジンにより情報量や表示順、検索速度などは異なるので、ユーザはどの検索サイトを利用するかを決め、その特性をつかんだり、いくつかの検索サイトを用いて結果を比較するなどを行わなければいけない。

Web サーバにファイルを置けば、Web を用いて情報発信が可能になる。情報は、一般に HTML を利用して記述する。Web ページの作成方

法についは後で学ぶ。

WWW を使ったコミュニケーション

WWW のはじめの頃は、WWW で情報を発信するのは Web ページを作る技術を持った人や企業・研究所などだけだったが、WWW 上で動作する**掲示板**や**チャット**などが増えると、これらに書き込む形で情報を交換することが広く行われるようになった。

最近では、個人が積極的に情報発信に参加するような Web の利用法が増えてきている。たとえば**ブログ** (blog) は、個人が Web 上で日記のような形の情報を手軽に公開するために作られたしくみである。また、**Wiki** は、Web 上でページの編集が可能で、グループで情報を共有したり、発信する情報の管理を行うことができる。**SNS**(Social Network Service) は、会員制をとることにより、不特定多数ではなく、限定された範囲で情報を発信するしくみである。多くの SNS では、会員になるために既に会員となっている人からの招待が必要である。

1.2　コンピュータネットワークの安全性

1.2.1　ネットワークに潜む危険性

ネットワークには多くのコンピュータが接続されており、非常に多くのさまざまな人がそれらを利用している。その中には悪意を持った人もいるため、そこからさまざまな危険性が生まれている。これらについて学んでおこう。

不正アクセス: **不正アクセス**とは、本来システムを利用できない者が不正にシステムを利用する行為をいう。システムを利用するための ID とパスワードを盗んで、他人の ID でシステムを利用したり、サーバプログラムのセキュリティホール (欠陥) を利用してシステムに侵入するものなどがある。

通信の覗き見: **覗き見**とは、通信路上の他者のデータを経由するサーバや通信機器で記録し、データを横取りする行為をいう。インターネット上の通常の通信は暗号化処理を行っていないので、ほとんどの通信内容は技術的には覗き見が可能である。このため、、電子

メールのメッセージ上や、Web ページのフォームに住所等の個人情報を書く場合は、常に覗き見の可能性があることに注意すべきである。特に、ネットショッピングなどでクレジットカードや銀行口座の番号を入力する場合は、通信が暗号化されたサイトであることを確認した上で利用することが重要である。

情報の改ざん: 通信中のデータに手を加え、全く違う通信をさせる行為を**改ざん**と呼ぶ。メールの中継ホストで内容を書き換えたりIP パケットの宛先を書き換えて送信者の意図しない計算機へデータを送るといったことが可能である。

情報の漏洩: 意図しない形で情報が他者に公開されてしまうことを**漏洩**という。インターネットを通じた情報の漏洩は、現在、大きな社会問題となっている。漏洩の原因はさまざまで、上で述べた不正アクセスや通信の覗き見もその一因となる。また、最近ではウィルスにより、意図しない情報や利用している操作画面の画像をインターネット上に送られるというトラブルが多発している。さらに、P2P を利用したファイル交換ソフトを通じて情報が送られることにより、漏洩した情報の回収が不可能になるケースも少なくない。

インターネット上の通信は一般的にさまざまなサーバやネットワーク機器を中継してデータのやりとりが行われている。したがって、中継する通信路を流れるデータは、それらの中継サーバや機器にアクセスできる者であれば、比較的簡単に覗き見ることが可能である。かつてインターネットは、ネットワーク利用者や管理者は不正なことを行わないという性善説で運営されていたが、その規模が非常に大きくなった現在では、ネットワーク上には悪意を持つユーザや管理者がいるということを常に意識しなければいけなくなったといえる。

1.2.2 ネットワークの安全性のための技術

ネットワークの安全性とは

安全性ないし**セキュリティ**(Security) とは、システム等が自分の意図したように利用でき (例:攻撃や妨害によって利用不能にならない)、意

図しない動作をしない (例:情報の漏洩などを起こさない) ことを表す概念である。

ネットワークの場合は、その通信線を切断したり電波障害を起こすなどの形での妨害を防ぐのは難しいが、一方でそのような妨害は発見と対処も容易なのであまり問題になることはない。むしろ、次のようなことが問題になりやすい:

- 知らないうちにネットワークを他人に使われてしまう。
- 知らないうちに機密の情報を他人に知られてしまう。

これは、インターネットに機器を接続することで、その機器がインターネットに自由にアクセスできるようになると同時に、インターネット上の誰もがその機器に自由にアクセスできるようになるという性質があるためである。

以下では前者の問題を防ぐ手段である認証とファイアウォール、後者の問題を防ぐ手段である暗号化について説明する。

認証

複数の利用者が 1 つのシステムを共用する場合は、各利用者に対して ID を発行し、**ID** と**パスワード**によって本人であるかを**認証**することが一般的である。

インターネット自体も、多数の利用が共用するシステムであるが、個々のパケットを認証するのは手間が掛かりすぎるため、認証は行わないのが普通である。このため、勝手にネットワークを使われてしまわないためには、ケーブルや機器に勝手に線をつながれないように注意する。

一方、無線 LAN の場合はケーブルを物理的に見張るわけに行かないので、何らかの認証を用いて不正に使用されることを防ぐのが普通である。無線 LAN では通信内容の盗聴を防ぐために WEP などの暗号機能が使われることが多いが、暗号機能を設定した無線 LAN ステーションは暗号キーを持たない人には利用できなくなるので、これが認証の役割も果たすことになる。

このほか、ネットワークにつながったコンピュータを外部から利用できるようにしている場合は、そのコンピュータを使う入口のところで ID とパスワードを用いた認証を行わせるのが普通である。

パスワードは、効果的に不正アクセスを防ぐためにも、容易に類推されるようなものであってはならない。たとえば、短いパスワードであれば、考えうる文字の組み合わせを片端から試せば、比較的短時間でパスワードが解析されてしまう。また、一般に用いられる言葉や名前をそのままパスワードにすると、辞書を利用した解析で破られる可能性が高くなる。これらの問題を避けるには、破られにくいパスワードを選ぶことと、パスワードを定期的に更新することが必要である。

一方で、破られにくいパスワードは憶えにくくなりがちなので、結果として正規の利用者がパスワードを忘れてしまうこともある。これを防ぐためには、たとえば自分がよく知っている全く別のコミュニティに属する 2 人 (幼ななじみとクラブの先輩など) の名前と好物など、利用者自身には憶えやすいが、すべてのものを同時に知っている他人はいないように、ものを複数組み合わせてパスワードを作るなどの工夫が必要である。

ファイアウォール

ファイアウォール (Firewall、防火壁) とは、もともとは火事の延焼を防ぐための壁のことであるが、ネットワークの場面では外部のネットワークとの間に設置して、不当なアクセスを防ぐ仕組みのことをいう。

インターネット自体はどのような人でも利用できるが、企業や組織のネットワーク、家庭内の LAN などはある一定の人 (学生/教員/職員、社員、家族) だけしか使わないし、使い方も限られている。そこで、インターネットとの出入口に「必要な使い方の通信だけを通す」機能を仕掛けることで、外部からの不正アクセスのための通信を遮断できるわけである。

実際のファイアウォールは、ルータの機能の一部として用意されいているものを使うこともあるし、パーソナルコンピュータの OS の機能に組み込まれているものを使うこともあるので、そのための目に見える特別な機器がないことも多い。

暗号化

盗聴からデータを保護するためには、通信データの暗号化が有効である。ここでは、インターネット上でよく利用される暗号化技術について

説明する。一般に、元の文 (**平文**、ひらぶん) を、定まった方法で別の文に変化させ、見ただけで内容がわからない文 (**暗号文**) にすることを**暗号化**と呼ぶ。逆に、暗号化された文を元に戻すことを**復号**とよぶ。

図 1.12: 文字の置き換えによる暗号の例

たとえば、

- HELLO

という文字列を暗号化するのに、図 1.12 の表にしたがって文字を置き換えるという方法を使ってみよう。つまり、A を X に置き換え、B を U に置き換え ···、というように文字を置き換えていく。すると、結果は以下のようになる:

- DYSSZ

この暗号を平文に戻すためには、図 1.12 の表を D → H、Y → E というように逆向きに使って文字を置き換えていけばよい。

この暗号化の方法でデータのやり取りをおこなうためには、あらかじめ “XUWKYBJDMOHSETZLPFVRANGQIC” という文字列を通信する双方で合意していればよい。これにより、もし通信が覗き見されたとしても、その内容が表を使って暗号化されているとわからなければ無意味な文字列に見え、内容が盗み見られることはない。[3]

一般に、暗号化・復号を行うために必要な情報を**鍵**とよぶ。ここでは、文字列 “XUWKYBJDMOHSETZLPFVRANGQIC” が鍵となる。暗号を用いたデータのやりとりでは、鍵の扱いが重要となる。ここで示した方法では暗号化と復号の際に同じ鍵を用いているが、このような暗号化の方法を**共通鍵暗号方式**と呼ぶ。共通鍵暗号方式では、通信する者どうしが他者には教えない**秘密鍵**を互い持ち合う。しかし、この方法では、秘密鍵をどのようにして互いに持ち合うか、また、やり取りする相手の数だけ秘密鍵を管理しなければならないなどの問題がある。

[3] アルファベット変換表を使った暗号は単純な方式で簡単に破れるてしまうので、実際の通信の暗号化には使われない。

公開鍵暗号方式

上に述べた共通鍵暗号方式の弱点を解消するために、暗号化と復号で異なる鍵を使用し、その一方を公開する**公開鍵暗号方式**が考案された。公開鍵暗号方式は、一人が多数の相手から暗号文を受け取り、復号したいときによく用いられる。

鍵の作成者は二つの鍵をペアで作成する。そのうちのひとつは本人のみが持ち、他人には知らせない秘密鍵とする。もう一つの鍵は、公に登録し、誰でも持つことが出来る**公開鍵**とする。ここで、公開鍵は暗号化のために用いられ、秘密鍵は復号のために用いられる。

今、鍵の作成者を A、A に対して暗号データを送ろうとしている人を B とすると、暗号データのやりとりは以下のような手順となる。

- A は自分の公開鍵を公に登録する
- B は A の公開鍵を取得し、それを用いて平文を暗号化する
- B は暗号文を A に送信する
- A は自分の秘密鍵を用いて暗号文を復号する

これにより、A は共通鍵暗号方式のように暗号通信をする相手の数だけ鍵を管理する必要がなくなる。このような方式は、ネットショッピングのように、不特定多数の顧客を相手にするような場合に、安全に顧客情報を送信してもらうための暗号通信には不可欠である。

1.3 Web サイトの作成

1.3.1 Web ページと HTML

HTML とマークアップ

我々が普段目にするテキスト情報 (文字の情報) は、単に文字が並んでいるだけということはなく、「見出し」「章・節」「段落」「脚注」「図版とその説明」などさまざまな要素が集まった構造を持っている。このような、構造を持ったテキストを**文書**と呼ぶ。本や雑誌、Web ページ、ワープロソフトで作成する書類などもこのような構造を持っており、文書の仲間だといえる。

文書の情報は、その内容となる文字に加えて「どこが見出し」「どこが段落」などの、構造をあらわす**付加情報**も含まれている。

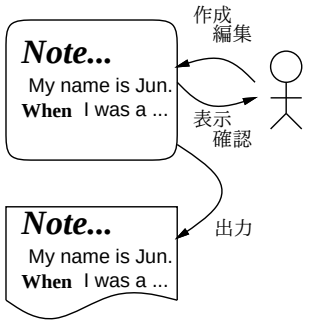


図 1.13: WYSIWYG(見たまま) 方式

ワープロソフトなどの場合は、最終的に印刷出力される見た目と同じものが画面に表示されていて、文字を打ち込んだり、文字や配置の変更を指示すると、それに応じて画面のようすも変化して結果を確認できる。このようなやり方を **WYSIWYG**(What You See Is What You Get、見たまま) 方式と呼ぶ (図 1.13)。

別の方法として、内容となる文字と一緒に特別な印を入れることで付加情報を表す方式がある。これを**マークアップ** (Markup、印つけ) 方式という。Web ページは、**HTML**(HyperText Markup Language) と呼ばれる規則でマークアップして作成する (図 1.14 左)。

Web ページの場合、使えるフォントや色などは読み手の環境によって変わってくるので、原理的に WYSIWYG 方式を使うことができない。また、画面が見られない人のために音声でページ内容を読み上げる場合などはそもそも「フォント」「色」が存在しない。このため HTML では、「見出し」「段落」「強調」などそれぞれの範囲の意味をマークアップで指定し、実際の見え方 (や読み上げ方) はブラウザに任せるようになっている (図 1.14 右)。[4]

HTML の書き方規則

HTML 文書は、冒頭に **DOCTYPE 宣言**を置くことにより、どの版ンの HTML であるかを明示することになっている。以下では HTML

[4]画面上で指示することで HTML のマークアップを生成してくれる「WYSIWYG ふう」作成ソフトも多数あるが、そこでの画面の見え方は「一例」に過ぎないことを意識しておく必要がある。

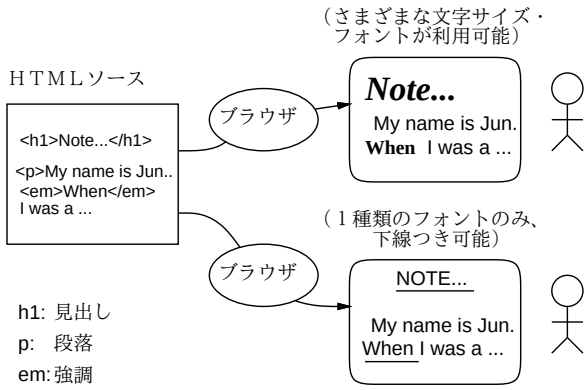


図 1.14: HTML とマークアップ

4.01 厳密版を用いることにして、次の DOCTYPE 宣言を置くことにする:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
```

これ以降には HTML の記述を行う。HTML では**タグ**と呼ばれる印 (マークアップ) で範囲を囲んで指定する形が基本となる。これを**要素**と呼ぶ。要素名には小文字を使うことにする。

- <要素名 追加指定…> … </要素名> — 開始タグと終了タグで囲む基本的な形の要素。
- <要素名 追加指定…> — 単独のタグ (タグ 1 つで 1 つの要素)。

開始タグ (と単独のタグ) には追加指定をつけることもある。タグで囲む要素の中に、さらに別の要素を入れられる。このとき、タグが互い違いになってはいけない:

```
○  <a> ... <b> ... </b> ... </a>
×  <a> ... <b> ... </a> ... </b>
```

タグに用いる「<」、「>」は特別な文字であり、そのままでは文書の内容として使うことができない。使いたい場合はそれぞれ「`<`」「`>`」と書く必要がある。また、このために「`&`」も特別な文字であり、文書の内容として使いたい場合は「`&`」と書く必要がある。[5]

[5]lt は less than(より小)、gt は gerater than(より大)、amp は ampersand(アンド記号) から来た名前。

基本的な HTML 文書

HTML 文書には最低限、次の要素が含まれている必要がある。

- <html>…</html> — HTML 文書全体をあらわす。この内側には head 要素と body 要素がこの順で入る。
- <head>…</head> — ヘッダ (HTML 文書自体に関する情報を入れる部分) をあらわす。
- <title>…</title> — HTML 文書のタイトルをあらわす。必ず head 要素の内側になければいけない。
- <body>…</body> — HTML 文書の本文 (ブラウザの窓内に表示される内容) をあらわす。

文書の内容をあらわす要素がないとサンプルが作れないので、とりあえず次の 2 つの要素を使うことにする。

- <h1>…</h1> — 大見出し。[6]
- … — 段落。HTML では普通の「地の文」も段落など何らかの要素の中に入れる必要がある。

次にごく簡単な HTML 文書の例を示す:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html>
<head>
<title>sample</title>
<head>
<body>
<h1>HTML のサンプル</h1>

<p>HTML では「&lt;」、「&gt;」で囲まれた「タグ」によって
文書をマークアップする。HTML の文書はブラウザによって
それぞれの環境に応じて整形・表示される。</p>
</body>
</html>
```

この HTML 文書をブラウザで表示したようすを図 1.15 に示す。ブラウザが h1 要素 (大見出し) を大きい文字で、段落を小さい文字で表示していることが分かる。また、ブラウザ窓の幅を変えた場合も、ブラウザが窓の幅に応じて内容を整形していることが分かる。

内容をあらわす要素をもう少しだけ学んでおく。

[6] <h1>〜</h1>から<h6>〜</h6>までの 6 レベルの要素があり、番号が大きいほど下の (小さいレベルの) 見出しとなる。

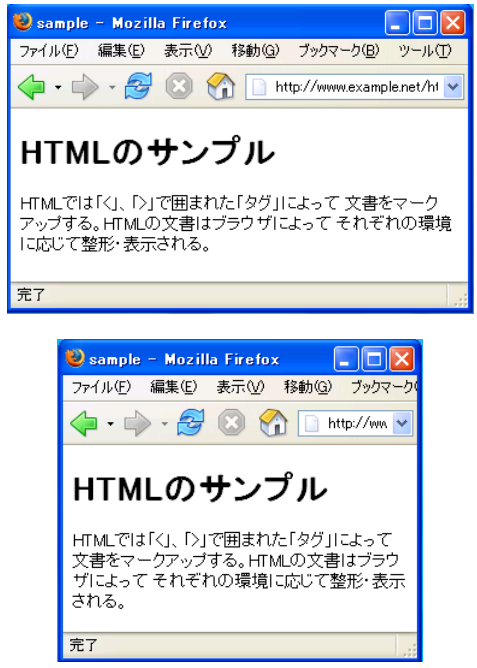


図 1.15: 簡単な HTML 文書の表示

- `<pre>…</pre>` — 整形ずみテキスト。この要素の内側は空白や改行が「そのまま」残される。
- `<hr>` — 区切り線 (単独のタグ)。内容の切れ目などをあらわす横線を引く。
- `<br>` — 強制改行 (単独のタグ)。段落などの内側はブラウザによって自動的に整形されるが、とくに改行したいところがあれば br 要素によって指定する。
- `<em>…</em>` — 強調。この要素の内側はイタリック体になるなど、目立つように表示される。

pre 要素や hr 要素は**ブロックレベル要素**と呼ばれ、段落と同列に扱う (段落の内側には入れられない)。段落 (p) や見出し (h1〜h6) もブロックレベル要素である。一方、br 要素や em 要素は**インライン要素**と呼ばれ、文字などと同列に扱う (必ず段落などの内側に入れる)。

これらを使った例も示しておく (図 1.16):

図 1.16: 簡単な HTML 文書の表示 (2)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html>
<head>
<title>sample</title>
<head>
<body>
<h1>HTML のサンプル (2)</h1>

<p>br 要素を使えば、任意の位置で<br>行を替え
させることができる。</p>
<hr>
<p>また、pre 要素を使えば、空白や改行をそのまま
表示させることができる。</p>
<pre>
  たとえば、
      この
          ように。
</pre>
</body>
</html>
```

演習 例題の HTML をそのままファイルに打ち込んで (ファイル名の末尾は「.html」にすること)、ブラウザで開いて表示させてみよ (文章の中身は変えてよい)。さらにタグと文章を増やしてみよ。窓の幅を変えたり、複数のブラウザでの表示を比較してみよ。

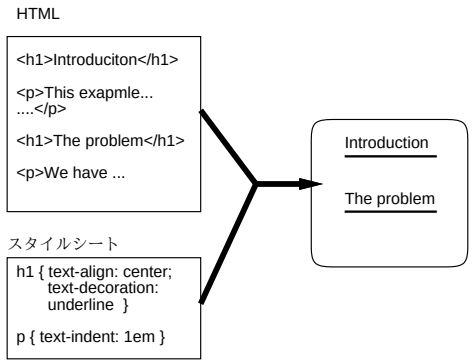


図 1.17: スタイルシートの原理

1.3.2 表現の指定とスタイルシート

HTML と表現の指定

ここまでに学んだように、HTML では「ここは段落」「ここは見出し」など、それぞれの範囲の「意味」を指定する。これに対し、文字や背景の色、段落の形など「表現 (見えかた)」を指定するのには、**スタイルシート**と呼ばれる機能を使う。

スタイルシートでは、個別の箇所ごとに色や文字サイズを指定するのではなく、「大見出しは茶色の文字で下線つき」「段落の冒頭は 1 文字下げて」など、要素の種別ごとにまとめて表現を指定する (図 1.17)。これにより、1 箇所の指定だけですべての要素に統一的な見え方を持たせることもできる (必要なら個別の箇所に対する指定も行える)。

以前の HTML では、色などを指定するのにもタグ (要素) を使用していた (現在はこのような要素は非推奨になっている)。しかし、強調のために赤字にしようとして「赤色にする」というタグを使ったとしても、カラーでない表示装置や音声ブラウザではどうにもならない。これに対し、em(強調) 要素でマークアップし、スタイルシートで em 要素を赤字に指定しておけば、カラーでない表示装置や音声ブラウザでもそれなりの (太字や大きめの声などの) 方法で強調を表すことができる。

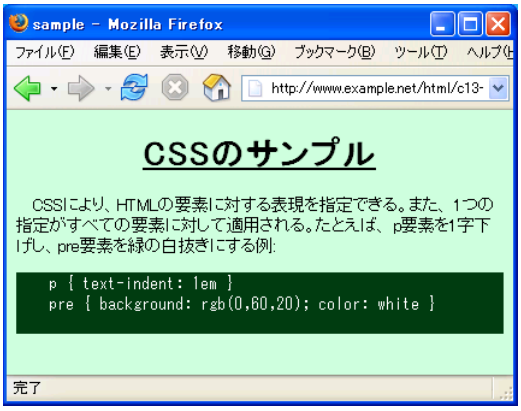


図 1.18: 簡単な CSS 指定の表示

CSS による表現の指定

以下では **CSS**(Cascading Style Sheet、直列スタイルシート) と呼ばれる記法で表現を指定していく。その指定は一般に次のような形をしている:

セレクタ { プロパティ: 値… ; プロパティ: 値… ; … }

セレクタは「何に対して」、プロパティは「何を」、値は「どう表現」を指定する。

値の指定方法として、「色」を指定する場合は、「*#rrggbb*」形式または「`rgb(`*r*`,`*g*`,`*b*`)`」形式で 24 ビットカラーを指定する。そのほか、`blue`、`red` などいくつかの色については直接名前で指定もできる。「長さ」を指定する場合は、「`mm`」「`cm`」など長さの単位をつける。長さの単位としてはほかに「`px`」(画面上のピクセル数)、「`em`」(小文字の「m」の幅)、「`ex`」(小文字の x の高さ) などの指定ができ、また「`30%`」などのように百分率を指定することもできる (ページ全体や指定対象の要素の外側要素の幅/ 高さに対する比率を指定することになる)。

CSS によるスタイル指定を HTML に適用させるのには、複数の方法があるが、ここでは HTML の style 要素を用いて HTML ファイルの中に埋め込む方法を説明する (表示のようすを図 1.18 に示す):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html>
<head>
<title>sample</title>
```

```
<style type="text/css">
body { background: rgb(200,255,220) }
h1 { text-align: center; text-decoration: underline }
p { text-indent: 1em }
pre { background: rgb(0,60,20); color: white }
</style>
<head>
<body>
<h1>CSS のサンプル</h1>

<p>CSS により、HTML の要素に対する表現を指定できる。また、
1 つの指定がすべての要素に対して適用される。たとえば、
p 要素を 1 字下げし、pre 要素を緑の白抜きにする例:</p>
<pre>
  p { text-indent: 1em }
  pre { background: rgb(0,60,20); color: white }
</pre>
</body>
</html>
```

このように、head 要素中に style 要素を含め、その内側に CSS 指定を書くことで、この HTML 内容に適用する CSS 指定を含めることができる。上の例題では次のプロパティを指定している:

- `color:` 色 — 文字の色を指定する。
- `background:` 色、`background:` `url(`画像ファイル`)` — 背景の色、または背景画像を指定する。
- `text-align:` 指定 — 文字の左/右/中央揃えを指定する。ここで「指定」としては `left`(左揃え)、`right`(右揃え)、`center`(中央揃え) が指定できる。
- `text-decoration:` 指定 — 文字飾りを指定する。ここで「指定」としては `underline`(下線)、`overline`(上線) `strike-through`(抹消線) などが指定できる。
- `text-indent:` 長さ — 段落 1 行目の字下げを指定する。

演習 先に作成したページに CSS の指定をつけてみよ。どのような表現が読みやすいか検討してみること。また、他人がつけた CSS 指定を (CSS の部分だけ) コピーしてきてみて、自分の指定とどちらがいいか検討してみよ。

1.3.3　HTML のさまざまな要素

箇条書き

文書を分かりやすく提示する手段の 1 つに、**箇条書き**がある。箇条書きでは、自分が提示したい分類ないし要素の枠組みごとに 1 つの項目を立てることで、これらの分類や要素をはっきりと読み手に示すことができる。

HTML の「番号なし」「番号つき」箇条書に用いる要素を示しておく:

- `<ul>…</ul>` — 番号なし箇条書き (unorderd list) 全体を表す要素。
- `<ol>…</ol>` — 番号つき箇条書き (orderd list) 全体を表す要素。
- `<li>…</li>` — 箇条書きの項目 (list item) を表す要素。ul 要素と ol 要素に共通で使う。

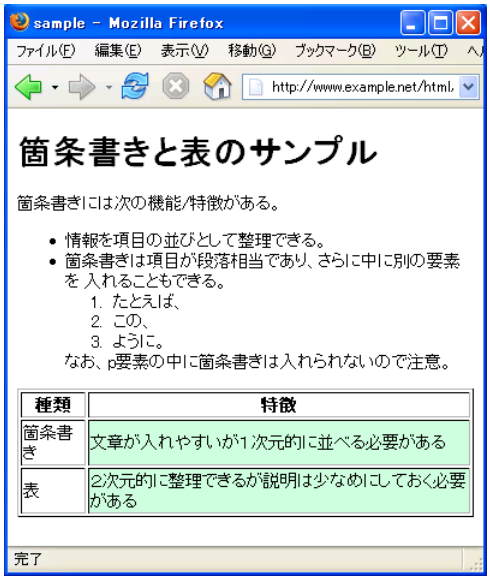


図 1.19: 箇条書と表の例

ul 要素、ol 要素の中には li 要素だけが入るが、li 要素の中にはさらに別の要素が入ることができる)。これらを使った例を示す (図 1.19、table 要素の表示も参考のために入れた):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html>
<head>
<title>sample</title>
<head>
<body>
<h1>箇条書きと表のサンプル</h1>
<p>箇条書きには次の機能/特徴がある。</p>
<ul>
 <li>情報を項目の並びとして整理できる。</li>
 <li>箇条書きは項目が段落相当であり、さらに中に別の要素を
  入れることもできる。
  <ol><li>たとえば、</li><li>この、</li><li>ように。</li></ol>
  なお、p 要素の中に箇条書きは入れられないので注意。</li>
</ul>
(途中略)
</body>
</html>
```

演習 自分の練習用ページに箇条書きを入れてみよ。また「定義型の箇条書き」「表」のための要素も調べて入れてみよ。

リンク

リンクは WWW におけるもっとも重要な機能の 1 つであり、多くのページで使われる。リンクは HTML では a 要素によって表す:

- `<a href="`URL`">`リンクテキスト`</a>` — リンクを表す。

「リンクテキスト」の部分は画面に特別な表現 (通常は青の下線つき) で表示され、そこ選択すると指定した URL に表示が切り替わる。a 要素はインライン要素であり、段落や箇条書き項目などの内側に含まれている必要がある。行き先としては、「`http:`」などから始まる URL 全体を指定する代わりに、ファイル名を指定することもできる (その場合、現在見ているページと同じ場所にあるファイルを表示する)。

WWW の初期においては、リンクテキストとして「ここ」などの指示語を指定することが流行したことがあったが、これは次のような理由からよくないとされている:

- ページを印刷したときに何を意味しているか分からなくなる。
- リンクを選択してみないと、行き先が何であるのか分からない。

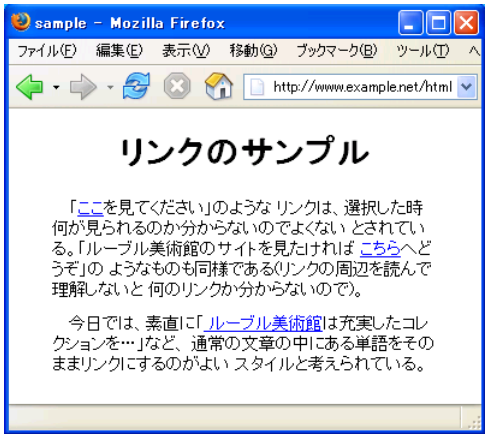


図 1.20: リンクの例

たとえば「ここ」リンクが多数並んだページを想像してみれば、その不便さが分かるはずである。今日では、リンクテキストはその行き先の内容を表す言葉をそのまま用いるのが好ましいとされている。これらの違いを示す例を挙げる (図 1.20):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html>
<head>
<title>sample</title>
<style type="text/css">
h1 { text-align: center }
p { text-indent: 1em; margin: 4px 1em; padding: 4px 1em }
</style>
<head>
<body>
<h1>リンクのサンプル</h1>

<p>「<a href="index.html">ここ</a>を見てください」のような
リンクは、選択した時何が見られるのか分からないのでよくない
とされている。「ルーブル美術館のサイトを見たければ
<a href="http://www.louvre.or.jp/">こちら</a>へどうぞ」の
ようなものも同様である (リンクの周辺を読んで理解しないと
何のリンクか分からないので)。</p>

<p>今日では、素直に「<a href="http://www.louvre.or.jp/">
ルーブル美術館</a>は充実したコレクションを…」など、
通常の文章の中にある単語をそのままリンクにするのがよい
スタイルと考えられている。</p>
```

```
</body>
</html>
```

演習 自分の練習用ページにリンクを入れてみよ。どのような入れ方が読みやすく使いやすいか検討してみよう。

画像の利用

画像は Web ページにインパクトを与え魅力的に見せる重要な手段であるが、それを濫用するとかえってページの情報を伝えにくくしてしまう。画像を使うときは、その画像が情報伝達のために有効である場面を見極め、節度を持って使うようにしたい。Web ページに画像入れる方法としては、次の 3 種類がある:

- リンク画像 — リンクの行き先として画像ファイルの URL を指定する。この場合、リンクが選択されるとブラウザはその画像を単独で表示する。
- 埋め込み画像 — ページの一部として画像を埋め込んで表示する。
- 背景画像 — ページ全体やその一部の要素の背景 (下地) として画像を表示する。

リンク画像は既に学んだ HTML の a 要素で指定できる。背景画像は CSS の `background:`プロパティの値として「`url(`URL`)`」という形で画像ファイルの URL を指定する。埋め込み画像は HTML の img 要素で指定する (これは閉じタグのない単独の要素):

- `<img src="`URL`" alt="`説明文`">` — 埋め込み画像を表す。`src` で画像ファイルの URL を指定する。`alt` の内容は、画像が表示できない環境で変わりに表示や読み上げに使われる。

img 要素はインライン要素であり、大きくても通常の文字 1 文字と同等に扱われる。このため、p 要素などの中に入れる必要があるが、他の文字と一緒に入れると img 要素の入る行だけ高さが高くなる。このため、1 個の p 要素の中に img 要素を単独で入れたり、前後に br 要素を入れて行変えするなどの工夫が必要になる。

以下に埋め込み画像と背景画像を使った例を示す (図 1.21)。この例の CSS には「`#`名前」という形のセレクタが現れているが、これは HTML 側で「`id="`名前`"`」という指定を持つ特定要素に対する指定を意味する。この方法で「この要素だけこのような表現にする」という指定が可能になる。

図 1.21: 埋め込み画像と背景画像の例

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html>
<head>
<title>sample</title>
<style type="text/css">
#p1 { text-align: center }
#p2 { background: url(img2.png) }
</style>
<head>
<body>
<h1>画像のサンプル</h1>

<p>ここでは少し大きな画像と背景用の画像を使います。</p>
<p id="p1"><img src="img1.png" alt="画像 1"><br>少し大きな画像</p>
<p id="p2">画像をページ全体や特定要素の背景として使うことも
できますが、薄い色でないとうるさくなります。</p>
</body>
</html>
```

演習 自分の練習用ページに画像を入れてみよ。どのような画像の使い方が効果的か考えてみよう。

第2章　情報とその活用

2.1　メディアとコミュニケーション

2.1.1　さまざまなメディア

情報とメディア

情報とは、何らかのことがらについての知らせである。たとえば、コインを投げて表か裏かが出たとして、その「表か裏か」ということを誰かに伝えれば、結果を知らなかった人が結果を知ることができる。

メディア (**媒体**) とは、情報を記録したり伝達する手段のことをいう。たとえば、「表か裏か」は葉書で伝えることも電話で伝えることもできる。この場合、葉書や電話がメディアということになる。情報自体には形がないため、文字でも音声でも伝えることができるわけである。

文字や音声が伝えている「表か裏か」を直接表すものは**データ**という。同じデータを 2 回受け取っても (既に結果を知っているのにもう 1 度教えて貰っても) 受け手の情報は増えない。まとめると次のことがわかる:

- 情報は形を持たない。
- 情報は複製でき、複製しても増えたり減ったりしない。

このような、形をもたず、いくらでも複製でき利用できるという性質が、今日のような情報の大量生産/大量消費を支えていると言える。

ところで、メディアは情報伝達の「手段」であるが、そこには情報を伝える「モノ (物理媒体)」も必要である。また、メディア上での情報の「表現」にも複数の選択肢があることが普通である (図 2.1)。いくつか具体例を見てみよう:

手段: 新聞、雑誌、本、テレビ放送、電話など。

モノ: 紙とインク、音、電波や電気信号など。

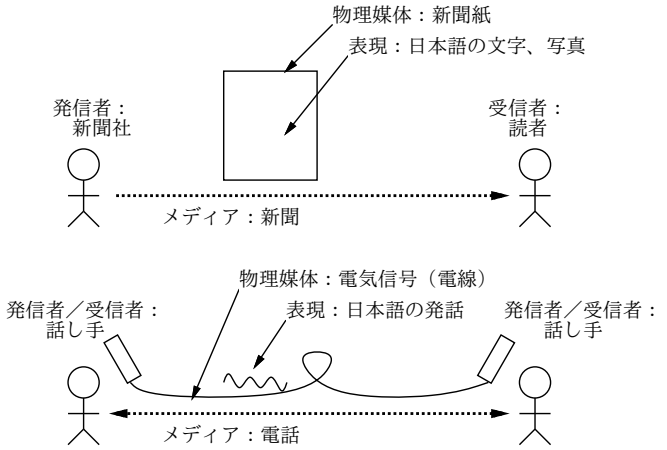


図 2.1: メディアとその要素

表現: 日本語の文字、日本語の発話、絵、写真、パントマイムなど。

今日では、コンピュータとネットワークの発達によって、これまでに無かったほど急速に新しい「ネットワーク上のメディア」が誕生し普及してきている。

たとえば、**電子メール**は誕生してからまだ 35 年ほどしか経っていないが、今では誰もが PC や携帯端末上でメールを使うようになった。また、**WWW**(World Wide Web — ネット経由で Web ページを見ることができるシステム) は誕生してからまだ 15 年ほどしか経っていないが、今日では WWW が多くの人にとってさまざまな情報を調べる主要な手段となった。[1]

メディアの分類基準

ここまでで、メディアのさまざまな例が出てきたが、メディアにはさまざまな性質ないし分類基準がある。具体的に見てみよう:

[1] 多くの人が「インターネットをする」という言い方をするが、それは実はインターネット上のサービスの 1 つである WWW を使っている、というのが正確である。インターネットは通信のための基盤であり、その上に WWW、電子メール、ネット放送、ネット電話などさまざまなサービス (ないし媒体) が提供されている。

片方向/双方向: **片方向のメディア**とは、情報の流れが一方通行のものを言う。新聞、テレビ、ラジオなどのマスメディアはおおむね片方向である。**双方向のメディア**とは、情報の流れが両方向にあるもので、電話などがあてはまる。

1 対 1/多対多/1 対多: **1 対 1 のメディア**とは、個人どうしでやりとりするもので、電話、私信、電子メールなどが代表的である。**多対多のメディア**とは多人数どうしがやりとりするもので、掲示板やチャットが相当する。**1 対多のメディア**の場合、「1」から「多」に情報が流れるものが多く、テレビ/ラジオ放送や WWW などが相当する。逆に「多」から「1」に情報が流れるものとしては、投書箱やその Web 版 (Web 上のアンケート) などが相当する。

個人メディア/マスメディア: **個人メディア**とは個人どうしが情報をやりとりするもので、会話、電話、私信、電子メールなどがあてまる。この場合当然、1 対 1 で双方向のメディアということになる。**マスメディア**とは多くの人に向けて一斉に情報を流すことができるようなものを言い、1 対多で片方向のメディアということになる。

実時間/蓄積型: **実時間メディア**とは、発信側が情報を発信しているその同じ時点で受信側が情報を受け取れるようなものをいい、テレビやラジオの生放送、電話、チャットなどがあてはまる。**蓄積型メディア**とは、送信側が情報をいったん蓄積 (保管・記録) して受信側が後でその情報を受け取るものをいい、新聞や雑誌、電子メールなどがあてはまる。

2.1.2 コミュニケーションとコミュニティ

コミュニケーション

一般に**コミュニケーション**とは、複数の人どうしが意思、感情、情報などを伝え合うことをいう。なお、新聞、テレビなど多くの人に一斉に情報を伝達するものを**マスコミュニケーション** (**マスコミ**) と呼ぶ。単にコミュニケーションといった場合は非常に広い範囲の概念が含まれるが、ここではさまざまなコミュニケーションのうちから、「主として情報機器を介した、人と人との意思疎通」を中心に取り上げる。

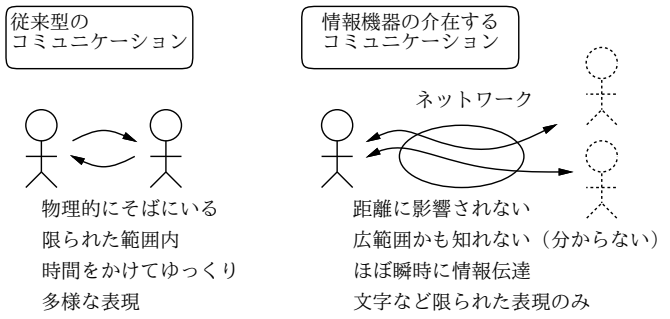


図 2.2: 新しい情報手段によるコミュニケーション

なぜ「情報機器を介した」が問題になるのか考えてみよう。情報機器を介した情報伝達には、従来の人力や物理的媒体を介した情報伝達と比較して、次のような特性がある (テレビ、ラジオ、ファクシミリ、電話なども情報機器であることに注意):

- 非常に短時間で遠距離まで/広範囲に情報を伝達できる
- 物理的に隔たった場所の間で情報を伝達できる
- 情報の発信者が誰か分からないままに情報を伝達できるものがある

さらに、今日のインターネットや WWW、そして携帯などモバイル機器の発達により、コンピュータネットワークを介した情報伝達では次のような新たな特性も加わっている。

- 国境や政治体制などの社会的障壁を越えた情報伝達が容易である
- 個人が容易に世界に向けた情報発信を行える

このような特性の多くは旧来の情報伝達手段になかったものであるか、またはできたとしてもその程度が大きく違っている。このため、新しい情報手段がもたらす情報伝達の速度、量、広範囲さ、およびその影響はわれわれの身体的感覚を大きく超えたものとなってきている (図 2.2)。

たとえば、自室で自分のブログに今日あったできごとを書き込むのは私的な行為のように感じられるかも知れないが、書き込んだ瞬間にその情報は世界中に向かって発信され、どこの誰が読んでもおかしくはない。しかしそのような意識をきちんと持ち続けている人は多くはない。それ

は、自室で行う作業は私的な行為であり、せいぜいよく知っている仲間うちにしか影響しないという以前からの感覚があるためである。

このため、われわれは新しい情報手段を使うときには「未知の世界」に立っているのであって、それをどのように使って行けばいいかについて、十分慎重でなければならない、ということを常に思い起こす必要がある。

もう 1 つ重要なこととして、自分が直接対面しているのが情報機器であったとしても、その向う側で自分が発信した情報を読んだり、自分に情報を送っているのは他の人であり、コミュニケーションの対象はあくまでも人間だということを忘れてはならない。

コミュニティ

コミュニケーションと密接な関係を持つ言葉として**コミュニティ**がある。コミュニティとはもともとは「共同体意識を持って共同生活を営む地域や集団」の意味であったが、現在ではそれを援用して、日常的にコミュニケーションを取り合う人たちの集まり、という意味でも使われる。

ここでもやはり、情報機器を介してコミュニケーションを取り合う集団 (**ネットコミュニティ**) が重要となる。このようなコミュニティには、それぞれの人の生活する場所や時間帯、年齢や職業などの社会階層にほとんど制約されず、世界中のどこからでも特定の話題に興味を持つという共通点さえあれば参加できるところが、極めて革新的である。これにより、従来であれば成立し得なかった、ごく限られた人しか関心を持たないような特定のテーマに関しても、コミュニティが成立し得るようになった。

一方で、このような新しいコミュニティでは、これまでのコミュニティにくらべていさかいや対立、他人を傷つける発言などが起きやすい面もある。

従来型のコミュニティではその構成員は同じ場所にいるので、それぞれの構成員が「どういう人か」についても予め互いに理解しているし、必要なだけ情報交換が行え、また交換される情報にも言葉以外に見ぶり、表情など多くのものが含まれている。これにより、行き違いがあっても比較的すみやかに修復が可能である。

しかし、新しいコミュニティでは構成員どうしのやりとりは情報機器を経由する限られたものだけなので、十分な情報が伝わらない面があり、

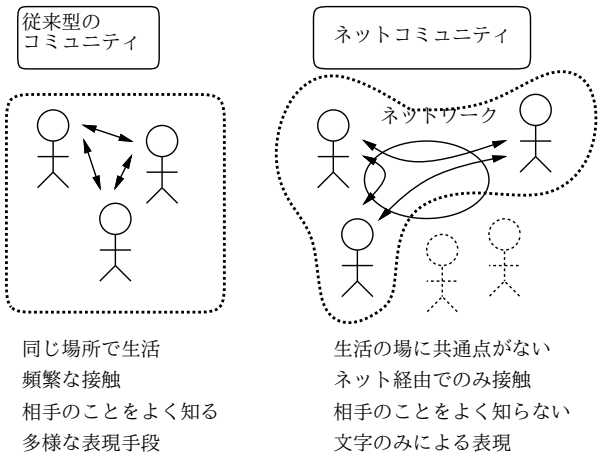


図 2.3: コミュニティの変革

さらに誤解や行き違いがあった場合にそれを正す機会も限られててしまう。また、自分がどのような人かをあまり明らかにしないまま参加することが可能であり、自分について虚偽のことがらを述べる参加者もいる。

ただし、このような側面があるからといって新しい種類のコミュニティが否定されるべきというわけではなく、そのような形ではじめて可能になる有益な情報交換も多く存在している。したがって、その特性をきちんと分かった上で、有益な情報交換が行えるように互いに注意しつつ活用していくことが望まれる。

2.2　情報の表現

2.2.1　アナログとディジタル

アナログ量とディジタル量

アナログ量 (**連続量**) とは、連続的に変化する値を表す量をいう。長さ、重さ、時間、温度、速度、力の強さなどはすべてアナログ量である。

ディジタル量 (**離散量**) とは、とびとびに変化する値を表す量をいう。ものの個数、組み合わせや場合の数など「数えられる」量がディジタル量に相当する。

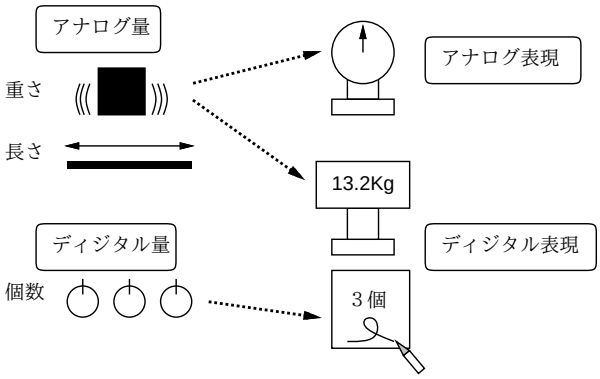


図 2.4: アナログとディジタル

量をあらわすときの表現にも、**アナログ表現**と**ディジタル表現**とがある。たとえばアナログ表現の時計や体重計では、針の位置が連続的に変化することで現在の時刻や体重を表す。一方、ディジタル表現の時計や体重計では、時刻や体重が数字で表される。

一般に、数字で量を表すことは、その数字の桁数で決まる最小単位(「1 秒」「0.1Kg」など) より細かい部分をまるめた「とびとびの」値を表すことになるので、すべてディジタル表現に相当する。

アナログ表現は、ぱっと見ておよそどれくらいか見てとりやすいという利点があるのに対し、ディジタル表現では数字で表示させるので値を正確に読み取るのに便利だという利点がある。ただし、ディジタル表現では最小の単位よりも細かい違いは読み取れない。

また、値を記録したり伝達するのにはアナログ表現よりもディジタル表現の方が優っている。たとえば、ものの長さを記録するのに、アナログ表現であれば紐などに印をつけて覚えることになるが、紐が伸びてしまったり印がかすれてしまうなどで、後で値を正確に再現できない可能性がある。また、遠くまでその情報を伝達するのも簡単ではない。

しかし、ものさしで長さを測って数字を書き留めておけば(ディジタル表現)、数字が読めなくならない限り値を再現するのも容易であるし、数字を読み上げたりして遠くまで伝達するのも簡単である。ただし、ディジタル表現にした時点でその最小単位より細かい情報は失われていることに注意しなければならない。

以下では簡単のため、「ディジタル表現によって表されている情報」のことを単に**ディジタル情報**と呼ぶことにする。コンピュータ内部ではすべての情報はディジタル表現によって表されている。これを短く書くと「コンピュータはディジタル情報を扱う」ということになる。

コンピュータとディジタル情報

ディジタル情報とは、別の見かたをすれば「いく通りかの場合のうちのどれか」という情報であると言える。たとえば、人の体重を「小数点以上 3 桁、小数点以下 2 桁の Kg 単位」で表すとすると、それは「000.00Kg ～999.99Kg までの 100,000 通りの場合のうちのどれか」という情報だと考えることができる。

ディジタル情報が一般に「N 通りのうちのどれか」だとすると、その最小単位は「2 通りのうちのどちらか」という情報だと考えることができる。これを「0/1 のどちらか」で表すこととし、「1 **ビット**の情報」と呼ぶ。たとえば、現在の天気を「雨が降っていない」「雨が降っている」の 2 通りの場合に分けたとすると、その情報をたとえば次のように 1 ビットの情報として表すことができる: [2][3]

ビット表現	意味
0	雨が降っていない
1	雨が降っている

1 ビットはディジタル情報の最小単位だが、複数のビットを並べた**ビット列**とすることで、より多くの情報を表現できる。たとえば、雨が降っている/いないでは大まかすぎるので、もっと詳しい情報として「晴れ」「曇」「雨」「雪」のどれであるかが知りたいとする。これは、たとえば次のように 2 ビットに対応させて表現できる。

[2] ビット (bit) は「2 進表現の 1 桁」(binary digit) から来ているが、「ちょっぴり」という意味の英語でもある。

[3] 前述の「既に知っていることを再度伝えられても情報は増えない」という観点から厳密に言えば「1 ビットのデータ」と呼ぶ方が正しい。また、知らないことであっても「雨が降ることが非常に少ない地方の天気」であれば、「雨が降っていない」という知らせには新たな価値がほとんどないから、雨と雨以外が半々の地域における「雨が降っていない」という知らせの場合に比べて、情報の量としては小さいものとなる、という考え方で**情報量**を測る理論もある。

ビット表現	意味
00	晴れ
01	曇
10	雨
11	雪

このように、ビット列の長さを 1 増やすと、表せる場合の数は 2 倍になり、一般に N ビットのビット列では 2^N 通りの場合を表すことができる。

そして、ディジタル情報とは「いく通りかの場合のうちのどれか」という情報なので、すべてのディジタル情報は (必要なだけの長さを決めることによって) ビット列で表すことができる。

コンピュータとはひらたくいえば、ビット列を蓄積/転送/加工するための装置であり、その機能によってあらゆるディジタル情報を取り扱うことができる (さらに、後で説明するように、人間の介在なしに自動的に処理を行えるという点も重要である)。

以下では、数値、文字、画像など代表的な種類の情報について、それをディジタル表現し、コンピュータ内部で扱う方法について見てみよう。

2.2.2 数の表現

十進表現と二進表現

コンピュータが作られた最初の目的は、人間に代わって文字通り「計算」を高速に/大量に/正確に行うことだった。このため、コンピュータで最初に扱われたデータの種類は**数値**だった。

数を表現する方法としては、漢数字 (一、二、三、四、…、九、十、十一、十二、…) もあるが、**アラビア数字** (0〜9 の数字) を用いた**位取り記法**が圧倒的に多く使われている。これは、位取り記法がなければ計算はほとんど不可能だからである。(たとえば千三百二十八から八百十三を「0〜9」で書き直さずに引き算してみれば分かる。)

我々が使う (**十進表現**ないし**十進法**の) 位取り記法では、数字として 0〜9 までの 10 種類ですべての数を書き表し、その値は桁が 1 増えるごとに十倍になる。たとえば「120」は「12」の十倍である。これは次のように説明できる: [4]

[4] 「x^y は「x の y 乗」を表す。x が何であっても、x^0 は 1 である。

$$1 \times 10^2 + 2 \times 10^1 + 0 \times 10^0$$
$$1 \times 10^1 + 2 \times 10^0$$

つまり、(十進表現の) 位取り記法で表された数は、左から順に $10^0 = 1$ 倍、$10^1 = 10$ 倍、$10^2 = 100$ 倍、…された値を表しているものとして扱われる。これによって、数字は 0〜9 までしかないのに、それを「並べる」ことでいくらでも大きな数が表せるわけである。

ところで、この「10」という値は特別ではなく、別の数を用いることもできる。この、位取りの基準となる数を**基数**と呼ぶ。我々が基数として「10」を使っている (十進表現を使っている) のは、単なる偶然 (指が 10 本あるから?) とされている。

これがもし「三進表現」であれば、数字として「0、1、2」の 3 種類を用い、1 桁右に行くごとに 3 倍の値を表すことになる。たとえば三進表現の「120」は次のように十進表現の「15」を表している (添字にかっこつきの数を書いて基数を表している)。

$$120_{(3)} = 1 \times 3^2 + 2 \times 3^1 + 0 \times 3^0 = 15_{(10)}$$

そして、コンピュータではおもに**二進表現** (ないし**二進法**) が使われる。これは、コンピュータの実現に使う電子回路では「電流が流れている/いない」「電圧がある/ ない」など 2 つの状態を持たせる回路が作りやすいためである。[5]

二進表現では、数値として「0、1」の 2 種類を用い、1 桁左に行くごとに 2 倍の数を表すことになる。たとえば「$1010_{(2)}$」は次のように十進表現の 10 を表す:

$$1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 8_{(10)} + 2_{(10)} = 10_{(10)}$$

二進表現と十進表現の相互変換

二進表現の原理が分かったところで、二進表現の表記を (普段使っている) 十進表現に変換する方法、逆に (普段使っている) 十進表現の表記を二進表現に変換する方法について学んでおこう。

[5] コンピュータの黎明期に、日本の独自技術として、3 状態を持つ回路素子を使ったコンピュータが作られたことがあり、そこでは三進表現が採用されていた。

まず二進十進変換から考える。一番簡単な方法としては、既に見てきたように、二進表現の数字に下から 1、2、4、8、…と位の数を割り当て、「1」のあるところの数を足すというものがある。たとえば $10010011_{(2)}$ を変換するところを見てみる (二進表現の各桁の値を左側に縦にならべた):

```
位の数   値
   128   1   128
    64   0
    32   0
    16   1    16
     8   0
     4   0
     2   1     2
     1   1     1
         -----
           147
```

逆に、十進二進変換はどうだろうか。これも、二進表現の原理どおりに考えるなら、上位から順に現在の値がその位の値より大きければその桁を 1 とし、現在の値から位の値を引いて行く方法がある。これを用いて $147_{(10)}$ を二進表現に変換するようすを示す:

```
位の数   桁の値  現在の値
                147
   128     1    147 - 128 = 19
    64     0
    32     0
    16     1     19 - 16 = 3
     8     0
     4     0
     2     1      3 - 2 = 1
     1     1      1 - 1 = 0
```

16 進表現

二進表現では任意のビット列を 0/1 の列として表すことができるが、文字が 0 と 1 しかないので大変長くなるという欠点がある。このため、代わりにビット列の表記には **16 進表現**が使われることが多い。

16 進表現では基数が 16 で、1 桁ごとに値が 16 倍になり、各桁の数字として 0、1、2、3、4、5、6、7、8、9、A、B、C、D、E、F の 16 種類を使う (16 進表現では数字が 16 個必要であり、0〜9 では足りないために、A〜F を数字として「借りて」きている)。

16 は 2^4 なので、16 進表現と二進表現の相互変換は簡単であり、二進表現を 4 ビットずつに区切ってそれぞれを次のように 0〜F に対応させるだけでよい:

二進	16 進
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

たとえば、二進表現の「1111 0000」は 16 進表現で表すと $F0_{(16)}$、二進表現の「1010 1011」は 16 進表現で表すと $AB_{(16)}$ となる。

2.2.3 文字の表現

文字と文字コード

われわれが日常接している情報の大部分はアナログ情報だといえる。たとえディジタル情報機器から出て来たものでも、映像も音楽もわれわれの目や耳に届くときにはアナログ情報の形になっている。

ただし、文字で書かれた情報だけはディジタル情報として受け取っている。というのは、文字の色や大きさなどは、書道などでその形を鑑賞するという特別な場合を除けば、読み取る内容には影響せず、まさに「知っている何千文字のうちのどれか」ということだけが問題になっているからである。

コンピュータで文字による情報 (**テキスト情報**) を扱うときもこれと同じ立場を取り、「どの文字」ということだけを取り出して扱う。具体的には、すべての文字に互いに異なる番号を割り当て、その番号を記録することでテキスト情報を記録する。この、文字に割り当てた番号のことを**文字コード**、あるひとまとまりの文字群とそれらへの文字コードの割り当て一式のことを**コード系**と呼ぶ。

コード系では、1 文字に何ビットを割り当てるかで収容できる文字数が決まってくる。たとえば、1 バイト (8 ビット) のコード系では $2^8 = 256$ までしか文字が収容できないため、漢字などを含む日本語は扱えない。日本語の文字を扱うときには、**JIS X 0208** と呼ばれる 2 バイトのコード系 (JIS 2 バイトコード) が標準的に使われている (**JIS** は日本工業規格— Japan Industrial Standard — の意味)。

英語圏の文字を扱う場合は、英大文字/英小文字/数字/記号が扱えれば済むので、これらを収容したコードである **ASCII** が標準的に使われている。ASCII コードを扱うときは、1 バイトにの 1 文字を入れて扱う。

なお、コンピュータの能力や記憶容量が小さかった時には、漢字を含む日本語を扱うのが難しかったため、ASCII(を多少変更したもの) にカタカナおよびカナ記号を加え 8 ビットに納めた **JIS X 0201** コード系 (JIS 8 ビットコード) が広く使われていた時期もあった。

符号化とファイル形式

文字コードはあくまでも「番号」であり、実際にその「番号」をコンピュータ上やコンピュータが扱うファイル上でどのようなビット列とし

下位4ビット＼上位4ビット	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	‘	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	’	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	NP	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

図 2.5: ASCII コード

て表現するかの規則は文字コードとは別に決められる。この規則を**符号化規則**という。

ASCII のような 7 ビットコードでは、1 バイト (8 ビット) に 1 文字ずつを入れればいいので符号化規則はごく単純でいいが、「行の切れ目」をどう表すかはシステムごとに変わって来ることがある。たとえば UNIX 系のシステムでは行の切れ目を ASCII の LF(改行) 文字で表すので、

```
My name is
Jun.
```

という 2 行のファイルは次のように符号化されている (図 2.5 と対照してみること):

```
4D 79 20 6E 61 6D 20 69 73 0A 4A 75 6E 2E 0A
```

日本語を含むファイルの場合、JIS 2 バイトコードと 1 バイトの ASCII の両方が使えるようにするため、これらを切り替え可能にして、その切り替え位置が分かるように符号化規則を決める必要がある。[6]

[6] 1 バイト/8 ビットコードを「半角」、2 バイト/16 ビットコードを「全角」と呼ぶこ

現在、JIS 2 バイトコードと 1 バイトの ASCII を切り替えて使うための符号化規則としては次の 3 種類が使われている:

- ISO-2022-JP — 単に「JIS コード」と呼ばれることもある。2 バイトコードと 1 バイトコードを特別な切り替え列と呼ばれる制御文字列で切り替え指示する。インターネット上のメール等ではこの符号化が多く使われている。[7]
- EUC-JP — 切り替え列を使う代わりに、各バイトの ASCII で使っていない 8 ビット目が「1」なら JIS 2 バイトコード、「0」なら ASCII として扱う。
- Shift JIS — パーソナルコンピュータ用の OS で多く使われている。EUC-JP に似ているが、8 ビットコード部分で ASCII だけでなく JIS 8 ビットコードのカナ (いわゆる半角カナ) も使えるように工夫した符号化規則。

以上 3 つは JIS 2 バイトコードに基づく符号化規則だったが、このほかに世界各国の文字コードを統一的に扱うことをめざした **Unicode** と呼ばれるコード系がある。Unicode をファイルに格納する場合は、**UTF-8** という符号化規則が多く使われる。

以上は符号化規則の説明だったが、さらに、行の切れ目の表し方も、オペレーティングシステムによって「LF」のみ (前記の例)、「CR」のみ、「CR LF」の列という 3 種類が使われている。

2.2.4 画像の表現

色の表現

絵、写真、図などの「見えるもの」の情報は「色」と「かたち」から成っている。まず色から考えてみよう。我々の目の中にある受光細胞には、赤、緑、青の 3 種類の光に主に感応するものがあり、その混ざり具合によってさまざまな色が感じられる。このため、この 3 つの色を「光の三原色」という。

ともあるが、本来文字コードは文字の表示幅とは無関係なのであまり適切な用語とは言えない。

[7] ISO-2022-JP 以外の符号化以外を扱えない電子メールソフトも使われているので、今のところ電子メールの送信には (受信側ととくに合意がある場合以外には) この符号化を使うのが無難である。

コンピュータで (明るさを含めた)「色」を扱うときには、この赤 (Red)、緑 (Green)、青 (Blue) の強さをそれぞれ数値として表し、その 3 者を合わせたものでさまざまな色を表す方法が多く使われる。これを 3 色の頭文字を取って **RGB カラーモデル**と呼ぶ。

その中でも多く使われるのが、RGB 各色の強さをそれぞれ 8 ビットの二進表現 (0〜255 の整数を表現可能) で表し、合計 24 ビットで色を表す方法で、これを特に **24 ビットカラー**と呼ぶ。24 ビットカラーの色を表記する時には、「`#`*rrggbb*」(*rr*、*gg*、*bb* は 16 進 2 桁で各色の強さを表す) または「`rgb(`*r*`,`*g*`,`*b*`)`」(*r*、*g*、*b* は 0〜255 の整数で各色の強さを表す) という書き方を用いることが多い (HTML と CSS で使われる記法)。

たとえば、`#FFFFFF` (`rgb(255,255,255)`) は、各色が最大に明るく光った状態であり、「白」を表す。`#000000` (`rgb(0,0,0)`) は各色がまったくない状態であり、「黒」を表す。`#FF0000` (`rgb(255,0,0)`) は、赤だけが最大、残り 2 色がない状態であり、「真っ赤」を表す。`#A0A0A0` (`rgb(160,160,160)`) は、各色が均等にかなり明るく光った状態であり、明るめの灰色を表す。`#A0A0C0` (`rgb(160,160,192)`) は、上記の灰色に少し青みがかった色を表す。

上で述べた方法では、複数の色を加えてまぜ合わせることでさまざまな色を作り出し、全部の色を加えると白になる。これを**加色混合**という。

これに対し、絵の具やインクを混ぜ合わせる場合は、それぞれのインクは特定の色の光を吸収することで (吸収されなかった光の) 色が見えるので、複数のインクを混ぜると吸収される色が多くなって暗くなり、すべてのインクをまぜると真っ黒になる。これを**減色混合**と呼ぶ。印刷などではシアン、マゼンタ、黄色の 3 種類を混ぜてさまざまな色をつくり出す (この他に黒のインクも使うことが多い)。

ピクセル画像とベクトル画像

画像をディジタル情報として扱う方法の 1 つは、画像全体を縦横のます目に細かく区切り、それぞれのます目の色を (たとえば 24 ビットカラーなどで) ディジタル表現することである。このような画像は、いわば多数の「色のついた点」が縦横に並んでできていると言える。この個々の点を**ピクセル**、このような形式の画像を**ピクセル画像**と呼ぶ (図 2.6 左)。

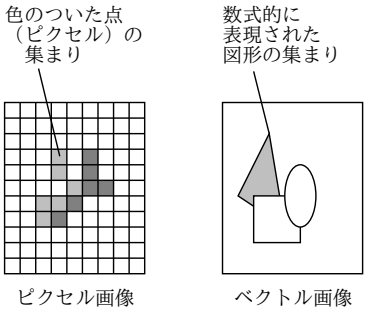


図 2.6: ピクセル画像とベクトル画像

ピクセル画像の利点は、コンピュータの画面やプリンタに出力可能なものであればすべて表現できるという点である。また、スキャナーやディジタルカメラで取り込んだ画像もピクセル画像である。

ピクセル画像の弱点は、拡大に弱いこと (個々のピクセルより細かい情報はないので、あまり拡大するとギザギザ状になったりモザイク状に見えてしまう)、加工が面倒なことなどである。

また、データサイズが大きくなりがちという弱点もあるが、これはファイルに格納するときに**圧縮**を行うことである程度克服可能である。ピクセル画像を格納するファイル形式としては、次の 3 種類が多くつかわれる、これらはいずれも Web ページ上でも利用できる。

- **GIF 形式** — 古くからあるファイル形式で最大 256 色までしか扱えないため、アイコンなど色数の限られた画像に適している。
- **JPEG 形式** — 24 ビットカラーを扱え、写真などの圧縮に適した圧縮方式と組み合わせて使われる。この圧縮方式は**損失のある圧縮**と呼ばれ、復元した時に完全に元の画像には戻らないが、その分圧縮率を高めることができる。圧縮率を選択することができ、圧縮率を高くするほどファイルが小さくなるが、戻した時の元の画像とのずれも大きくなる。
- **PNG 形式** — 24 ビットカラーを扱え、**損失のない圧縮** (復元したときに元の画像と同じに戻る) を備えている。比較的新しいファイル形式。

もう 1 つの画像形式として、画像をさまざまな幾何学的図形の集まりとして扱うものがあり、**ベクトル画像**と呼ばれる (図 2.6 右)。ベクト

ル画像では、個々の図形は数式的に表されているため、拡大しても粗くなったりギザギザに見えるようなことはない。また、画像の複雑さにもよるが、ベクトル画像のデータサイズはピクセル画像より小さいことが多い。また、個々の図形が個別に保持されているため、後から個々の図形の位置、大きさ、色などを変更しやすい場合が多い。

一方、写真のような外部から取り込んだ画像はもともと図形の集まりではないため、ベクトル画像に変換するのは難しい。また、描画ソフトでもピクセル画像向けのソフトでよく使われる「にじみ」「ぼかし」などの操作はピクセル単位で行うので、ベクトル画像向けのソフトでは通常使うことができない。

2.3　情報機器と人間の接点

2.3.1　コンピュータと人間の情報処理

人間の情報処理

われわれ人間が持つ情報処理能力を整理して考えて見よう (図 2.7)。われわれの身体には複数の**感覚器**が備わっており、そこから絶えず多様な情報を取り入れている。これを**五感**と呼ぶ:

- **視覚** — 感覚器は**目**であり、色、形、明るさなどの情報を取り入れることができる。視覚は人間にとって多くの情報を取り入れる主要な感覚器であり、他の動物と比べても、微妙な色や形の違いを見分けられるという点で優っている。また、眼球が前向きに 2 つ並んでいることから、視差を利用して奥行きを判断できるという特徴もある。
- **聴覚** — 感覚器は**耳**であり、音の強さと高さや音色 (さまざまな**周波数**の音の混ざり具合い) の情報を取り入れることができる。人間の聴覚は周波数がおよそ 20Hz〜20kHz の範囲の音を聞き取ることができ、言葉を聞き取る上で重要な役割を果たしているが、人間の聴覚が他の動物と比べてとりわけ優れているというわけではない。
- **味覚** — 感覚器は**舌**であり、甘い、辛い、苦いなどいく種類かの味を感じ分けることができる。

- **嗅覚** — 感覚器は**鼻**であり、さまざまな匂いを感じとることができる。味覚や嗅覚はその仕組みがまだよく分かっていないところが多い。たとえば「いい味」「いい匂い」を数量的に定義する方法は確立していない。
- **触覚** — 感覚器は**皮膚**であり、温暖や力、痛みなどを感じることができる。目が見えない人が手の触覚を利用して点字を読む様子を見ると、訓練次第でかなり細かい情報が取得可能なことが分かる。

これら五感に入らない「虫の知らせ」「予感」のようなものを**第六感**と呼ぶこともあるが、実際にそのようなものが存在するかどうかはまだはっきりしていない。

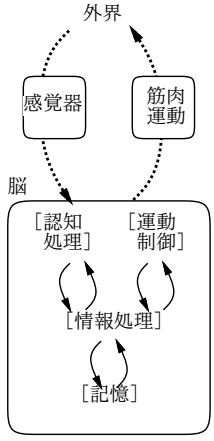


図 2.7: 人間による情報処理

人間が感覚器から取り込んだ情報は、**神経系**の働きによって**脳**に伝達される。脳は多数の**神経細胞**が集まった人間の情報処理の中枢であり、ここで感覚器からの情報が解釈される (認知処理)。

脳では外部から情報を取り入れ解釈するだけでなく、情報を短期間/長期間にわたって**記憶**する能力や、これらの情報をあらためて検討して新たな情報を作り出す能力、さまざまな問題について考えて答えを出す能力、芸術や文化などを味わう能力、喜怒哀楽などの感情を作り出す能力、自我の意識 (自分は誰であるという継続性のある感覚) を作り出す能力など、非常に多くの情報処理能力を備えている。

脳で加工された情報を外にあらわすときには、脳の運動制御機能から発した指令が神経系を通って人体の各所にある筋肉に伝わり、その運動を通じて外界に働きかける。たとえば言葉は喉 (声帯) と舌と口の筋肉の働きによって発することができるし、顔の表情、まばたき、手や足の動きなどもすべてそれぞれの場所に備わった筋肉の運動によって作り出される。

人間とコンピュータの情報処理の特徴

コンピュータの情報処理機構も、大まかな構造では人間によく似ている (図 2.8)。すなわち、マウス、キーボードなどの**入力装置**から読み込まれた情報は本体内の **CPU**(Centeral Processing Unit — 中央処理装置) によって処理され、その結果がディスプレイなどの**出力装置**を通じて提示される。CPU 内部には情報を加工する演算機能や、どのように加工を行うかを制御する制御機能が備わっている。情報の保持には CPU に隣接した**主記憶**を用いる (CPU 内部には少量の情報だけが保持できる)。

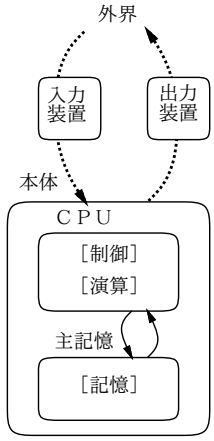


図 2.8: コンピュータによる情報処理

人間と比べたコンピュータの特徴は、ディジタル表現された情報を扱い、厳密に動作内容が定まった命令を高速に実行するように作られているので、大量の情報を決まった形で処理することを得意とすることである。

コンピュータで曖昧さや確からしさを扱うことは可能だが、その場合はたとえば「曖昧さ」「確からしさ」を数値化してその数値を計算していく形で扱うことが必要になる。さらにその場合「曖昧さ」をどう計算するかという手順について厳密に定めておく必要があり、その手順をうまく定めるのは簡単ではない。

一方、人間の情報処理は脳に集まっている神経細胞の回路をもとにしており、神経細胞どうしの情報伝達機構には「信号の強さ」が組み込まれていて、これを土台にして連想や感覚といった曖昧さを含む情報をうまく扱えるようになっている。

その反面、厳密に決まった手順を実行するような場合には、電子回路と比べて神経細胞の情報処理はずっと遅いうえ、人間の情報処理機構の特徴とは相反する作業であるため効率が悪く、「飽きる」「間違える」「疲れる」などの現象のため、正しく処理ができないことがある。

わが国のコンピュータ科学の先駆者である高橋秀俊は人間の特性を次のように記している (高橋秀俊、「数理と現象」、岩波書店、1957):

- 人間は気まぐれである。
- 人間はなまけものである。
- 人間は不注意である。
- 人間は単調を嫌う。
- 人間は根気がない。
- 人間はのろまである。
- 人間は論理的思考が弱い。
- 人間は何をするかわからない。

これを読むと人間は欠点だらけのようであるが、このような特性があるからこそ、人間には大きな創造性や柔軟性が備わっているとも言える。

一方で、これらの欠点が望ましくないところでは、いくら「注意しろ、努力しろ」と言っても、これらは人間に本来備わった特性であるので、精神論では克服できない。そうではなくて、今日ではコンピュータの能力を活用して人間の動作を補うことが可能になってきているので、その方向を目指すべきである。このように、それぞれが得意とする形の情報処理を分担し、人間がうまくコンピュータを道具として使いこなしてゆくことが、今後いっそう重要になって行くだろう。

人間とコンピュータの対話

先に、人間は感覚器から情報を取り込み、脳で処理し、運動を通じて出力すること、コンピュータは入力装置から情報を取り込み、CPU で処理し、出力装置から出力することを学んだ。では、人間がコンピュータを使っている状況では何が起こっているだろうか。

答えは、コンピュータの出力装置の出力 (おもに画面表示) は人間の感覚器を経て脳に入り、人間が判断したことは運動による入力装置 (おもにキーボードやマウス) の操作を経てコンピュータに入力され、CPU がそれを処理した結果がまた出力される、という形のサイクルになるということである。つまり、コンピュータと人間が合わさって 1 つのサイクルを形成する (図 2.9)。

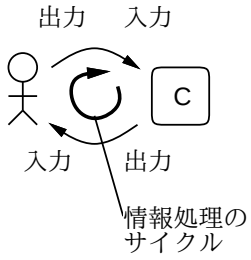


図 2.9: 人間とコンピュータの対話

このサイクルは「適切な位置までマウスポインタを移動していく」というごく短い周期のものであることもあるし、数値を打ち込んで計算を指示し、コンピュータが結果を表示したらそれを検討して別の数値で計算しなおしてみる、というような長い周期のものであることもある。

今日では多くの人がコンピュータの前で長い時間作業するので、人間にとって使いやすいようにコンピュータの (つまりソフトウェアの) 動きが構築されていることは大切である。このような研究分野全般のことを、「人間とコンピュータの対話」(**HCI**、Human-Computer Interaction) 分野と呼んでいる。これに対し、**ユーザインタフェース**という言葉は、「コンピュータのうちで人間とのやりとりに関わる部分」というやや限定された意味で使われる。

2.3.2 ユーザインタフェース

ユーザインタフェースの多様性

人間とコンピュータがやりとりする方式として、過去においては、キーボードから文字の並びによる**コマンド**を打ち込み、コンピュータからの応答も文字で示される方式が主流だった (**CUI** — Character/Command User Interface、図 2.10 左)。今日では、マウスなどの**ポインティングデバイス**で画面に表示されている操作対象や実行すべきコマンドを選んで指示する方式 (**GUI** — Graphical User Interface、図 2.10 右) が広く使われている。

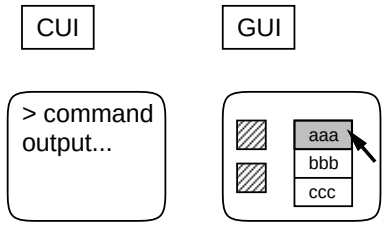


図 2.10: CUI と GUI

このような変化は、GUI に必要な入出力装置が廉価に作れるようになり普及したことと、GUI の方がコマンドを記憶する必要性が小さく、初めての人にも分かりやすいことが大きな要因であるが、CUI は使われなくなったかというと、そうではない。専門家が多様な処理を大量に指示する必要があるような場面では、効率よく複雑な指示を出すために、今でも CUI が多く使われている。

また、携帯電話などの場合には、画面が小さいことと、片手で効率よく操作できることから、キー (ボタン) でメニューを選択するインタフェースが主に使われている。このように、ユーザインタフェースにはさまざまな方式があり、それぞれの特性によって適する場面や適さない場面がある。

「使いやすさ」とは

多くの人が長時間コンピュータを使う今日では、ソフトウェアのユーザインタフェースの出来不出来は人間の作業効率に大きく影響する要因であり、このため、「使いやすい」ユーザインタフェースを持つことは重要だといえる。

では**使いやすい**とは具体的にはどういうことだろうか? たとえば「使っていて気持ちがいい」「好みである」というのは誰でも思い付く基準だが、このような**主観的評価**は人によって違って来るため、できるだけ数量化できる、客観的な基準を用いたい。

たとえば、ある作業を決めて、違ったユーザインタフェースを持つ 2 つのソフトウェアでその作業をこなすのにかかる時間を計測して比較すれば、どちらがどれだけ速く作業をこなせるかが数量的に示せる。しかし、作業が速くこなせてもすぐ疲れてしまうようなユーザインタフェースでは、たくさんの作業をこなすのには向かないことになる。

このように、ユーザインタフェースに限らずどのようなものでも、評価を行う基準は複数あり、どれか 1 つの基準だけで善し悪しが決まることはあまりない。

以下にユーザインタフェースの評価基準として考えられるものを挙げておく (網羅的ではない):

- 主観的評価 — 好みである、使っていて気持ちがいいなど
- 所要時間 — 作業を決めて、その作業をこなすのにかかる時間を計測
- 間違えにくさ — 作業を決めて、その作業をこなす間にどれくらい間違いをおかすかを計測
- 学びやすさ — はじめて使う人が一定の習熟状態になるまでにかかる時間
- 忘れにくさ — しばらく使って使い方を覚えたあと、使わなくなってもどのくらい使い方を忘れずに覚えているか
- 認知負荷 — 人間にとってどれくらい負担になるか。たとえば一定時間使ったあとでフリッカーテスト (光のちらつきがどれくらい認識できるかで疲労度を測定する手法であり、測定器も市販されている) などで疲れ方を測ることが考えられる。

これらの基準によっては相反する場合もある。

たとえば、メニューを用いたインタフェースは「どんな操作があるか」覚えていなくてもメニューを表示させてみれば分かるため、はじめての人でもすぐ使えるようになる。しかし、習熟した人にとっては、いちいちメニューを出して選択するのはわずらわしいことが多い。

一方、コマンドを打ち込むようなインタフェースはコマンドを覚えていなければ使えないので、はじめての人には敷居が高い。しかしコマンドを覚えてしまえば、それを打ち込むだけで済むので、習熟した人には楽に操作できる。

アクセシビリティ

使いやすさの基準一般とは別に、ハンディキャップを持った人でもうまく使えるようにユーザインタフェースを設計することが必要である。このような性質を**アクセシビリティ**と呼ぶ。ここでいう「ハンディキャップを持った人」とは、身体的ハンディキャップの他にも、歳をとって目や耳が悪くなった人、使っている機器の表示能力などに制約がある人まで含めた広い概念である。

一般に、コンピュータからの出力は視覚と聴覚によるものが主流だが、目が悪い人のためには「文字を大きくする」、目がまったく見えない人のためには「内容を音声で読み上げる」「多数の小さいピンの凹凸を制御して点字や形の情報を伝える」などの方式がある。また、耳が聞こえない人のためには音で知らせる代わりに画面表示を活用して知らせる (警告ブザーの代わりに画面が一瞬明滅するなど) 方法がある。

コンピュータへの入力についても、マウスがうまく操作できない人のために矢印キーなどを活用する、キーボードそのものに触るのが困難な人のために音声、息、まばたきなどを使って意図を伝えるなどの方法がある。

このように、コンピュータの入出力において、1 つの手段だけに限らず複数の手段を活用するインタフェースを**マルチモーダル**なインタフェースと呼ぶ。

ユーザビリティ工学

ユーザビリティ工学 (Usability Engineering) とは、ユーザインタフェース等を「使いやすく」設計/制作するための手法や体系を研究する分野

のことである。

これは、ソフトウェアだけに限らず、ウェブページなどにも適用される (**Web ユーザビリティ**)。とくに商用のウェブページでは、ユーザにスムーズに見てもらえることが利益につながるため、さまざまな検討がなされている。たとえば、配色やデザインを工夫すること、ページをいくつかの領域 (タイトル部分、サイドバー部分、本体部分) に分けて構成すること、ページ間の移動のしかたを統一し使いやすくすること、などが代表的である。

ソフトウェアでも Web ページでも、ユーザがそのソフトウェアや Web ページを使うときの作業内容の流れを図に描いて、スムーズに作業が行えるかを検討するなどの方法が使われる。また、制作物全体を作る前に、そのユーザインタフェース部分だけを**プロトタイプ** (試作模型) として作って検討することも行われる。

これらの設計全体を通して「ユーザに十分な情報を提供する」「余分な操作や無駄な後戻りを避ける」「メッセージは分かりやすく誤解しにくくする」などの指針を設けてチェックすることも重要である。

制作物が完成に近づいたら、想定される代表的なユーザ層を集めて日常生活の場で実際に使ってもらう**試用テスト** (Usage Test)、またはそれが難しければ実験室などの環境で一定時間試してもらう**ユーザビリティテスト** (Usability Test) などを行い、問題をチェックすることも行われる。

2.4　Web サイトの設計と制作

2.4.1　Web サイトの構築プロセス

Web サイトと構築プロセスの必要性

Web サイトとは、ひとまとまりの内容を公開するため Web ページの集まりをいう。Yahoo! Japan のような巨大なポータルも Web サイトであるし、あなたが練習のために作る数ページのページ群も Web サイトである (したがって、大きな Web サイトは多数の子サイト、孫サイトの集まりになっているはずである)。

ここまでで Web ページ制作の基盤技術である HTML と CSS について学んだが、以下では実際にどのようにして Web サイトやそれを構成

する個々の Web ページを設計し制作して行くかについて学ぼう。

ここでは Web サイトの構築を一種の問題解決と捉え、適切な問題解決プロセスを通じてサイトを設計/制作していくことにする。これにより、設計の見落としやその他の間違いのためうまく機能しないサイトを作ってしまう可能性を小さくできる。

我々は問題解決プロセスを一般に「問題の認識/同定/記述」「問題の分析」「解放の設計/実現」「検証/反復」のステップで捉えて来たが、ここではこの各ステップを次のものにそれぞれ対応させる:

- コンセプトデザイン — Web サイトが目的とする情報伝達の内容を具体的に定め、明文化する。
- サイトデザイン — コンセプトをうまく伝達するようなサイトの構成/構造の設計や、そこに含まれる各ページのデザイン (ページに含まれる領域とその配置、配色の決定) を行う。
- ページ設計/制作 — 個々のページの内容を決め、必要な素材を収集したりコンテンツの文書を書き、HTML+CSS により各ページを制作する。
- サイトの検証 — 制作したサイトの内容をチェックし、直すべき点があればその段階まで戻って修正する。

以下ではこ記のステップに従い、具体的な制作プロセスを経験していく。

チーム作業の概要

ここではさらに、複数人の**チーム**から成るプロセスを前提として実習ベースで進めることで、チームによる問題解決プロセスの要点や進め方についても並行して具体的に学んで行くことにしよう。

チームによるプロセスでは、とくに次のような点が重要となる:

- 役割分担の明確化 — ある作業を誰がやるか明確になっていないと、その作業は責任を持って行われないかも知れないし、重複して行われてしまうかもしれない。そのような状態で適切な成果物が得られるとは期待できない。なお、あるプロセスを決めれば、そこで必要な役割も決まる。作業の規模やチームの人数によっては、1 つの役割を複数の人が共同で担ったり、逆に一人が複数の役割を兼任することもある。

- 文書化と記録 — 各作業の成果がきちんと文書化されないと、成果をプロセスの次の段階に引き継ぎ、最終的な成果物まで結実させることができない。また、各作業の担当者、経過、起こったことなどがきちんと記録されていかないと、問題が起きた時にその原因を究明し、プロセスを見直して改善して行くことができない。プロセスを改善し、作業効率や成果物の品質を高めて行くことができなければ、チームはいつまでも「初心者」レベルに留まることになってしまう。
- 検証と修正 — どんなに優れた腕前を持つ人でも、作業のすべてを最初から完璧にこなすことは期待できない (人間は必ず間違いをおかす)。各段階の中間成果物/最終成果物いずれについても、それぞれが作られたらすぐに検証を行い、その品質が適切かどうかチェックする。また、そこで問題があれば手直しを行い、修正したものについて再度検証を行う。この過程は修正が不要との結論が出るまで繰り返し行われる必要がある。

これらの各点の具体例については、次節以降でプロセスの各段階ごとに例を挙げて行くことにする。なお、チームによる実習でなく一人で作業する場合でも、(役割分担については一人ですべての役を兼ねるとしても) 作業内容の明文化や記録などは同様に行うこと。

演習 Web サイト設計/制作の演習を行うために、数名規模のチームを編成する。役割として当面、リーダと記録係を決める。リーダはプロセスの進行を管理する。記録係は作業ごとに交替していってもよい。

2.4.2　設計段階

コンセプトデザイン

コンセプトデザインの目的は「何を目的とするどのようなサイトを作るか」を決めることである。以後の作業はすべて、ここで決まったことがらを実現することを目標として進めることになるので、コンセプトデザインを的確に行うことは重要である。

コンセプトデザインで決めることがらとしては、次のものがある:

チーム/文書番号:	かわせみチーム、K001
フォーム名:	コンセプトデザイン会議記録
作業日時:	2006.8.9 14:00-18:00 (理科実験室)
参加者名/記録係:	鈴木、佐藤、中村、田中 (記録: 田中)
(ブレーンストーミング) ・体育祭 Web — 体育祭のようすを記録し紹介するサイト。本番だけでなく、実行委員会の準備のようすや競技者の練習のようすなど知られざる努力も知ってもらう。(中村案) ・学年紹介 — 私達の学年がどんな学年か、どんな人がいるか、学校行事のときにどんな事件 (?) やできごとがあったかを紹介。(鈴木案) ・修学旅行 Web — 京都修学旅行に備えて、行ってみたらいいと思うところを調査して紹介する。(田中案)	
(検討分析) ・修学旅行 Web は実用性があるという点ではいい。(佐藤) ・修学旅行は画像を雑誌や Web から借用するのではまずい。よそへのリンクはあるとしても文字ばかりの内容になってしまうのでは。(鈴木) ・昨年京都に家族旅行に行った時の写真がいくらかあるが。(田中) ・体育祭は例年盛り上がるし、自分達の学年の記録を残す点で賛成。(鈴木) ・ちょうど制作期間中にあるのでデジカメ写真を十分入れられる。(田中) ・あの忙しい競技中にまんべんなく取材や撮影ができるか? (佐藤) ・競技進行ごとに担当を割り振って制作とかもちょっと難しそう。(田中) ・学年紹介であれば、テーマ (行事、人物) ごとに分担すれば形にしやすい。(佐藤) ・体育祭もその 1 つとして入れれば盛り込めるし目玉にできる。(鈴木) ・以上の議論の結論として、学年紹介をテーマとし、行事に体育祭も含めることとした。	

図 2.11: ブレーンストーミングの記録フォーム

- Web サイトのタイトル — サイトのコンセプトを的確に表し、親しみやすく覚えやすいタイトルを考えてつける。
- **コンセプト** — どのような情報を伝達することを目的とするかを定める。
- **ターゲット** — 情報を伝達する相手 (読み手) としてどのような層を想定するかを定める。

コンセプトで「どのような情報」と言う意味は、何の情報を伝達するのかという点に加えて、情報の詳細さ、情報の流れ方 (単方向/双方向/読み手どうしの交流など)、フォーマル/カジュアルさ、マルチメディアの利用など、サイトを特徴づけるものをすべて含む。

ターゲットを特定のものに絞るほど、サイトの設計は容易になる。たとえば「クラスメート」がターゲットであれば、どのような背景知識を持っていて何に関心があるかはおおよそ分かり、それに沿って設計を進められる。逆に、ターゲットを広くすると設計は難しい。たとえば「すべての人」とすると、小学生、お年寄り、年配の男性、若い女性などすべてを含むことになるが、これらすべてに興味を持ってもらえるような情報やその提示方法をうまく考えることは非常に大変である。

コンセプトを決めるにあたっては、**ブレーンストーミング**が多く使われる。ブレーンストーミングとは、参加者が自由な発想でさまざまなアイデアを出し、それを記録していく作業を言う。このとき、誰かが出したアイデアに対して批評を述べることは禁止する。内容について議論するのは後の段階の作業であり、アイデアを出している最中に内容について議論し始めるとそれ以上新しいアイデアが出にくくなってしまう。

ひととおりアイデアが出尽くしたら、それぞれの特徴、利点、欠点などを検討して整理しておく。後の参照のため、たとえば図 2.11 のような形で、文書化しておくのがよい。サイトのコンセプトの場合は、アイデアの中からどれか 1 つを選んで制作することにしてもよいし、複数を組み合わせることも考えられる。ブレーンストーミングの結果に基づき、コンセプトを取りまとめる。その内容も図 2.12 のように文書化しておく。

演習 チームで作成するサイトのコンセプトデザインを行う。まずブレーンストーミングを行い、その結果を整理する。それをもとに、皆が協力して作成できるようなサイトのコンセプトを決定する。これらの結果は図 2.11、図 2.12 のような形で文書化しておく。

チーム/文書番号:	かわせみチーム、K002
フォーム名:	コンセプト (2006.8.9 版)
作業日時:	2006.8.9 14:00-18:00 (理科実験室)
参加者名/記録係:	鈴木、佐藤、中村、田中 (記録: 田中)

Web サイトタイトル:

「第一高校 57 期の記録」

コンセプト:

・私達の学年も今はこうして忙しく毎日を過ごしているけれど、卒業すれば離れ離れになって互いのことを忘れていく。今のうちに、どんな有名 (?) 人がいたとか、どんな行事ではどんなことがあったとか、どの部活はどんな感じだったとか、記録できることを残しておきたい。卒業まではまだ間があるけど、卒業アルバムの Web 版で担当者次第でもっと生の話まで書けるというイメージ。

ターゲット:

・基本的に「内輪ネタ」で我々の学年同期が一番見ると思われる。ただしその前後の学年が見たり、また我々がよその知合いに高校のことを説明するのに見てもらったりもすると思われる。だから基本的に我々の世代 (高校生)。ただし卒業後 (大学生、就職後?) に見ても恥ずかしくないようにはしておきたい。

図 2.12: コンセプトの記録フォーム

サイトデザイン

サイトデザインの目的は、コンセプトを実際にサイトとして実現する上で、どのようなイメージのページを、どのような構造 (つながり方) で配置するのがよいかを決めることである。コンセプトが明確でも、そのコンセプトをうまく表すようなサイトデザインがなければ、読み手にうまくコンセプトを伝えることができない。

サイトデザイン作業の成果物 (設計文書群) 一式を**スタイルガイド**と呼び、これが以後そのサイトの作成に当たって必ず守るべき「憲法」となる。その中には少なくとも次のものが含まれる:

- 想定される読み手 (ターゲット) と、読み手がサイト内容を活用でき (アクセシビリティ)、また快適に利用できる (ユーザビリティ)

ために守るべき基準 (想定画面サイズ、色づかい、文字サイズ、各ページに必ず入れる情報、使ってよい/使わない技法など)。[8]

- 上記の指針にしたがった、サイトの基本構造の設計。
- 上記の指針にしたがった、ページデザインのテンプレート (ひな型)。

サイトのページ構成の基本的なものとしては**線形構造**と**階層構造** (木構造) とがあり (図 2.13)、これらのどちらかを選択するか (やや大きなサイトの場合) うまく組み合わせて全体の構造を作るのが普通である。

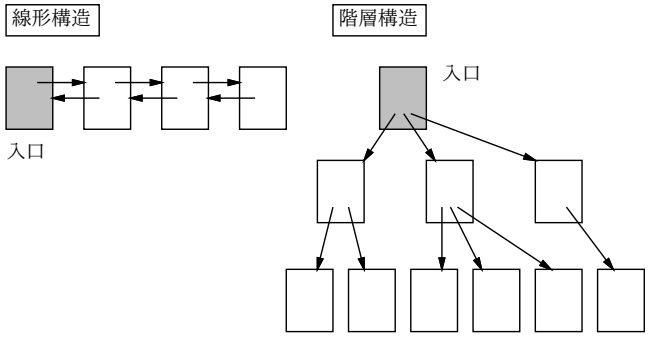


図 2.13: 線形構造と階層構造

線形構造とは、ページが直線的に並んだものであり、構造が単純で分かりやすい。また、各ページを順番に見て行くのに適している。途中のページに急いで行きたい場合にも、ページがあまり多くなければ、目次や一覧を用意することで対応できる。しかし、ページが非常に多くなると、目次や一覧を見て行きたいページを決めるのが困難になるため、階層構造を用いる方がよい。

階層構造とは、トップページ→大分類→中分類→小分類、のように枝分かれした構造であり、分類を順番にたどっていくことで目的のページに到達できるので、ページの数が多い場合でも対応できる。その反面、すべてのページを順に見て行くことはやりにくい。

これらの両方を組み合わせて、階層構造の「一番下」のページを線形構造で結んで順番に見て行けるようにすることもある (図 2.14 上)。この場合は、「一番下」以外の中間段階にあるページは目次的な内容だけを含ませる方が (順番に見て行く時には現れないので) よいかも知れない。

[8] アクセシビリティ、ユーザビリティの概念については第 1 章を参照のこと。

さらにサイトが大規模になってきた場合は、個々の「まとまり」を線形構造のページ群であらわし、どのページ群を選ぶかはトップページから階層構造に従ってたどっていく (図 2.14 下) などの方法が使われる。

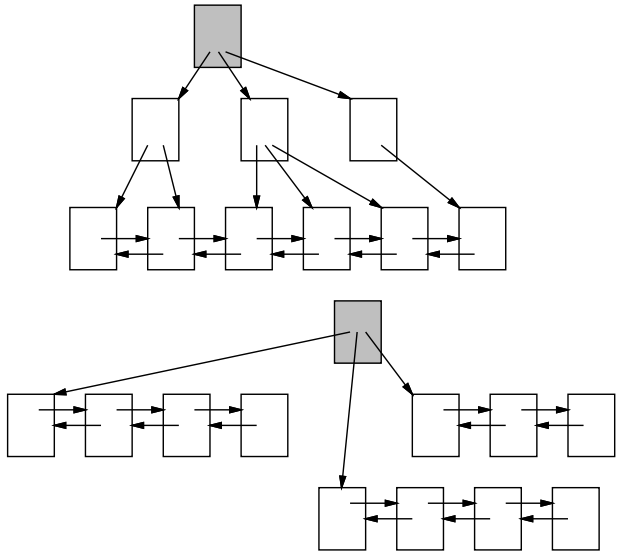


図 2.14: 線形構造と階層構造の組み合わせ

演習 チームで作成するサイトの構造を話し合って決める。どのような意見があってどのような理由から決めたかも適切な用紙を用意して記録しておくこと。

ページデザイン

ページデザインはサイトの Web ページが従う共通のデザインを決める作業であり、サイトデザインの一環として行われる。小規模なサイトの場合は、ページのデザインは 1 種類にしてどのページもそのデザインに従うことで、サイトとしての統一性を持たせる。大規模なサイトの場合は、その中を複数の領域やページの用途によって分け、それぞれに違ったページデザインを採用することもある。

いずれの場合も、さらにトップページだけは別のデザインにすることもある。これは、トップページが目だってすぐに覚えてもらえるようにす

る目的もあるが、大規模なサイトの場合は「最近のニュース」「よく使う機能やページへの近道」など多くの情報を盛り込む必要があって専用のデザインにすることもある。

一般のページについては、その内容には最低限、次の部分が含まれているはずである:

- バナー — ページの先頭部分にあり、そのページのタイトルを明確に示す。
- ナビゲーション領域 — 線形構造、階層構造などページの主たる構造に沿った移動を行うためのリンクを**ナビゲーションリンク**と呼ぶ。ナビゲーションリンクや、その他の (本文に埋まっている以外の) リンクを分かりやすく集めた領域をナビゲーション領域と呼ぶ。
- 本体 — 個々のページごとの内容を収める領域。

このほか、サイトの特性に応じてツールバー (そのサイト固有の機能を直接呼び出せるような領域)、コラム (主要なニュースなどを付加的に表示しておく領域)、その他の領域を持たせることもある。

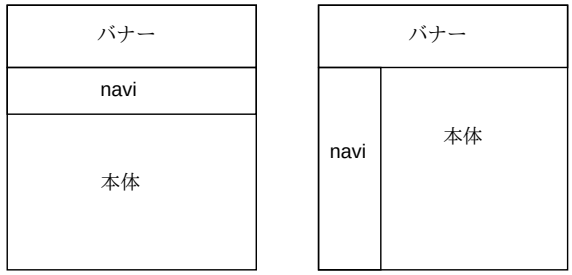


図 2.15: ページの中のレイアウト

ページに含める領域が決まったら、それらをどのようにページ内に配置するかもデザインしていく。たとえば図 2.15 左のように上からバナー、ナビゲーション、本体の順に並べることもできる。この方法は HTML の指定は簡単だが、バナーと本体の間にナビゲーション領域がはさまってじゃまに感じるかも知れない。図 2.15 右のように「サイドバー」にナビゲーションを入れるレイアウトであれば、上記のような問題がなく、ナビゲーション領域に目次や追加情報へのリンクなど多くの内容を含ま

せても読みにくくならない。このほか、サイドバーを右側に置いたり、左右両方に置いたり、ページ下端にも別の領域を配置するなど、さまざまなレイアウトが工夫されている。

ページデザインではレイアウト以外に配色も重要な要素である。たとえばパステルカラー、ビビッド (鮮やか) カラー、暖色系、寒色系などにより、優しい感じ、情熱的な感じ、暖かい感じ、涼し気な感じを出すことができる。色の選択はとくに、コンセプトの表現手段としても重要である。実際にどのような配置と色がいいかは、描画ソフトを使って複数の候補を「画面の絵を描いてみる」ことで試すとよい。色を調整したり各領域の大きさを調整するには、描画ソフトの機能がうまく利用できる。

図 2.16: テンプレートを表示してみる

最終的な候補が決まったら、今度は HTML と CSS を使って「とりあえずの」文字を入れながらページの「ひな型」(**テンプレート**) を作成する。図 2.16 の HTML と CSS を以下に示す:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html>
<head>
<title>sample</title>
<style type="text/css">
```

```
body { margin: 0px; background: rgb(0,60,0) }
#header  { height: 80px; border: green inset 6px;
  background: rgb(100,255,100) }
#thispage { float: left }
#sidebar { position: absolute; top: 92px; left: 0px;
  width: 90px; padding: 4px; color: white }
#sidebar a { color: white }
#main { margin-top: 0px; margin-left: 100px;
  padding: 4px; background: rgb(120,255,180) }
h1 { text-align: center; clear: both }
</style>
</head>
<body>
<div id="header">
<div id="thispage">page 1</div>
<h1>ページタイトル</h1>
</div>
<div id="sidebar">
← Prev | <a href="page02.html">Next →</a>
<hr>
page 1<br>
<a href="page2.html">page 2</a><br>
<a href="page3.html">page 3</a><br>
<a href="page4.html">page 4</a><br>
<a href="page5.html">page 5</a>
<hr>
<a href="http://example.com/">外部</a>
</div>
<div id="main">
<p>冒頭の文章。冒頭の文章。冒頭の文章。冒頭の文章。</p>

<h2>本体内見出し</h2>

<p>本文。本文。本文。本文。本文。本文。本文。本文。
本文。本文。本文。本文。本文。本文。本文。本文。</p>

<h2>本体内見出し</h2>

<p>本文。本文。本文。本文。本文。本文。本文。本文。
本文。本文。本文。本文。本文。本文。本文。本文。</p>
</div>
</body>
</html>
```

このようなレイアウトを実現する場合には、それぞれの領域を「id="固有名"」の指定をつけた div 要素として囲み、CSS 側でも「#固有名」と

いうセレクタを使ってその div 要素だけを対象としたスタイルをつける。ここではさらに、領域の外周のあき (margin) や内側のあき (padding) を指定し、サイドバーは絶対位置指定 (`position: absolute`) を指定して上端 (top)、左端 (left) の位置を指定することで、図 2.15 右のような配置を作り出している。

演習 チームで作成するサイトのページデザインを決めてみる。まず各自でコンセプトに従った案を描画ソフトで作って提案し、それぞれが自分の案を提示して説明した後、話し合いか投票で決める。作成した案は保存しておくこと。

2.4.3 制作/検証段階

ページ設計/制作

サイトデザインの段階でおおよそどのようなページをどのような構造でつなげるかは決まっているはずだが、具体的な個々のページが何であるかまでは決めていないかも知れない。ここではそれらの具体的内容を決め、担当や分担を決める。

続いて、その役割分担にしたがって素材 (絵や図や写真など) を収集/作成し、内容となる本文を打ち込んでファイルに保存しておく。用意した素材はいきなり HTML に入れるのではなく、素材のままの状態で一式を保存しておくのがよい。そうすれば後でテンプレートを取り替えるときに、素材を新しいテンプレートに挿入することで新しいページが作れる。

素材がすべて集まったら、ページごとに空っぽのテンプレートに素材を挿入することで、各ページを制作する。これらの作業全体を通じて、誰がどういう作業をしたか、その時どんな問題があったかなども記録しておくのがよい (あとで問題をふりかえったり、やり直すときに必要な情報となる)。

演習 実際にサイトの各ページについて素材を収集し、制作しなさい。分担のしかたは、ページごとに担当を割り当てるようにしてもよいし、素材や作業の種類 (写真、イラスト、文章、HTML) ごとに担当を分けてもよい。作業経過の記録も適切な用紙を用意してきちんと行うこと。

サイトの検証と保守

サイトが一通り完成したら、その内容についてこれでよいかどうか、検証を行う。

このとき、ただ漫然と見ているだけでは問題に気がつきにくい。このため、「何をチェックするか」を挙げたリスト (**チェックリスト**) を用意し、それと照合しながらチェックを行うことが多い。チェックリストの例を示しておく:

- すべてのページに適切なタイトルがついているか
- ページに含まれるすべてのリンクは正しい行き先になっているか
- ナビゲーションリンクの量は十分か (行きたいところに簡単に行けるか)
- ページのすべての文字は読める大きさや色づかいになっているか
- 画像が多すぎたり文字が多すぎたりしていないか
- 本文の内容がタイトルと合ったものになっているか

また、制作した本人達の場合、その内容を熟知しすぎていて問題に気がつかないこともある。このため、部外者に頼んでサイトを見てもらったり、サイト内にある特定の情報を指定して見つけてもらったりして、サイトが誰にでもうまく使えることを確認することも重要である。

問題があって修正する場合は、誰が担当となってどういう理由で何をどう修正したかをすべて記録する。修正作業は、サイトが完成して公開した後も問題に気がつくごとに続けて行く必要がある。これを**保守**と呼ぶ。

演習 作成したサイトについて、チェックリストによるチェックを行い、問題点があれば修正せよ。チェックリストとしてどのような項目が必要かも検討してみること。修正する場合はその修正内容をきちんと記録して残すこと。

第3章　コンピュータと情報

3.1　プログラミング入門

3.1.1　ドリトル言語によるプログラミング体験

ドリトル言語とその処理系

コンピュータがどのように動作するかは、すべて**プログラム**によって決められている。普段我々はオペレーティングシステム、ワープロソフトなど予め用意されているプログラムを使っているが、自分でもプログラムを書いてみることによって、コンピュータの動作原理をよりよく理解することができる。

プログラムを書くときの「書き方の規則」(どのように文字を並べて命令を組み立てるか、具体的にどのような命令があるかなど) のことを**プログラミング言語**という。プログラミング言語にはさまざまなものがあるが、ここではじめて「手順的な自動処理」を体験してみるのに適した言語として「**ドリトル**」という言語を取り上げ、具体的に学んで行く。[1]

プログラミング言語を実際にコンピュータで動かすためには、その言語による記述を読み込み、CPU が実行する動作に変換するようなソフトウェアが必要である。このようなソフトウェアを**プログラミング言語処理系** (ないし単に処理系) と呼ぶ。

ドリトルの処理系を起動したときの様子を図 3.1 に示す。処理系の画面は上部のタブにより次の 2 つの状態に切り替えられる:

- 編集画面 — プログラムを打ち込んだり修正したりする画面。プログラムは文字によって記述されるので、この画面ではテキストエディタやワープロソフトなどと同じような操作が行える。

[1] 詳しくは `http://dolittle.eplang.jp/` 参照。処理系やチュートリアル、マニュアルもここから取得できる。

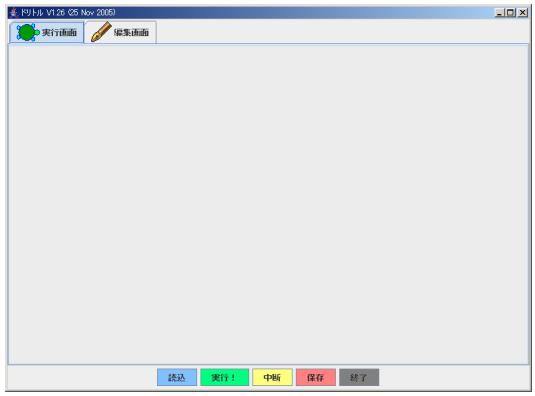


図 3.1: ドリトルの起動画面

- 実行画面 — プログラムを実行するときの画面。ドリトルでは命令によって絵を表示したり動かしたりできるので、この画面では絵を表示するための領域が見えている。

プログラムの入力と実行

プログラムを入力したり編集するときは、編集画面に切り替えて作業する (図 3.2)。簡単なプログラムとして、次の 1 行を打ち込んでみる:

```
タートル！作る。
```

プログラムを入力したら、画面下の “実行！” ボタンを押す。すると、自動的に実行画面に切り替わり、図 3.3 のように画面にカメが現れる。これでもちゃんと (非常に短いけど) プログラムを実行し、その結果として画面にカメが現れたことに注意。

オブジェクトと命令

ドリトルでは、プログラムが扱うさまざまな「もの」(データ等) を**オブジェクト**と呼ぶ。オブジェクトには次のような「呼びかける」書き方によってさまざまな指示を与えることができる:

```
オブジェクト！命令。
```

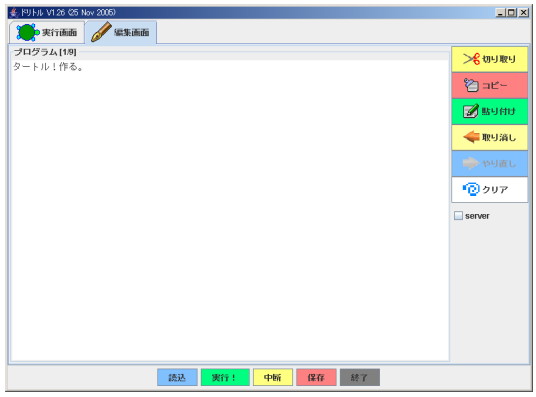


図 3.2: ドリトルの編集画面

図 3.3: 実行結果

先のプログラムではオブジェクトは「タートル」であり、これは “作る” という命令により、画面上に図形を描く機能を持つ「もの」(ペン) を作り出すことができる。

作ったペンにさらに命令を送りたいが、それには、呼びかけるための名前が必要である。プログラムを修正して、オブジェクトに “カメ太” という名前を付けてみる:

```
カメ太＝タートル！作る。
```

そして次の行を追加し、カメ太に「100 歩」歩く命令を送ってみる:

```
カメ太！１００　歩く。
```

「100」は歩く長さ、「歩く」は命令という別々のものなので、これらの間には空白を入れて分ける必要があることに注意。これを実行すると、図 3.4 のようにカメ太が歩いて画面に線が引かれる。

演習 ドリトル処理系を起動し、「線を引く」プログラムを動かしてみよ。
数値を変更すると線の長さが変わることを確認せよ。

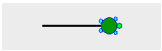


図 3.4: 実行結果

3.1.2　プログラムの組み立て

命令の連続

プログラムにより多くの仕事をさせるためには、複数の命令を実行させる必要がある。ドリトルでは、1 つの「もの」に複数の命令を続けて送ることができる。2 行目を修正して、次のように長くしてみる:

```
カメ太！１００　歩く　９０　左回り　１００　歩く。
```

編集画面で図 3.5 のプログラムを入力して実行すると、実行画面には図 3.6 のような線が描かれる。

図 3.5: 編集画面のプログラム

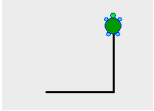


図 3.6: 線が描かれた

命令の繰り返し

上では 1 つの「命令文」で複数の命令を実行させていたが、さらに「命令文」を複数並べることで命令を増やして行くことができる。タートルに対する命令を増やしていくことで、画面にいろいろな図形を描く

ことができる。たとえば、次のプログラムを実行すると、画面に四角形が描かれる:

```
カメ太！１００　歩く　９０　左回り。
カメ太！１００　歩く　９０　左回り。
カメ太！１００　歩く　９０　左回り。
カメ太！１００　歩く　９０　左回り。
```

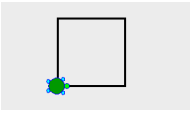


図 3.7: 四角形

四角形のプログラムをよく見ると、同じ命令を何度も実行していることがわかる。このようなときは、“繰り返す” を使い、プログラムを簡単にすることができる。

具体的には、繰り返したいプログラムを “「 」” で囲み、囲んだもの (これも「もの」になる) に対して “○回 繰り返す” という命令を送ることで、その囲んだものが指定した回数だけ実行される:

```
「カメ太！１００　歩く　９０　左回り」!4 繰り返す。
```

プログラムの実行結果は図3.7 と同じである。このように、ある動作を行うプログラムはいく通りもの書き方で書くことができる。では、どのような書き方がよいのだろうか。それは、人間にとって読みやすく、書きやすく、理解しやすいものがよい、といえる。では、上のプログラムではどうだろうか。繰り返しの概念が分からない人にとっては、1 番目の方が分かりやすいかも知れない。しかし繰り返しが分かってしまえば、2 番目の方が短く、簡潔で、理解しやすいかも知れない。

演習 「歩く」と「右回り」「左回り」を使っていろいろな図形を描いてみよ。様子が分かったら、最初に描く図形を計画して、その計画通りの図形が描けるようにプログラムを作成してみよ。

3.2　対話的プログラム

3.2.1　ボタンを使ったプログラム

対話的プログラムと GUI 部品

プログラムには、実行開始したら計算や制御などの処理を行い、人間がそこについていないことを前提とするものもあるが、人間がその場についていて使うことを前提としたプログラムもある。そのようなプログラムを**対話的プログラム**と呼ぶ。ワープロソフト、表計算ソフト、描画ソフトなどは、人間が操作しつつ文書や計算や描画を組み立てて行くので、どれも対話的プログラムである。

先の四角形を描くプログラムは、実行すると自動的にすべてが実行され、終わってしまうので、途中で人間が介在するところはない。つまり、対話的プログラムではなかった。

今度は対話的プログラムの例として、押しボタンを持ったプログラムを作ってみる。なぜボタンかというと、プログラムの実行中に人間がこのボタンを押すことによってプログラムの動作を調整できるようにするためである。

このように、対話的プログラムでは何らかの方法で実行中にプログラムが人間から情報を受け取ることが必要である。今日のプログラムでは、**GUI 部品**と呼ばれるさまざまな部品を使うことでこれを行うのが一般的である。GUI 部品の例としては、たとえば次のようなものがある:

- **ボタン** (押しボタン) — 押すと何らかの動作をする
- **フィールド** (入力欄) — 文字や数値を打ち込むことができる
- **スライダー** (スライドレバー) — 連続的な値を設定できる
- **メニュー** (選択メニュー) — 複数の選択肢から 1 つを選ぶ

ボタンを使った例題

では、実行中に人間との間で情報をやり取りする、つまり対話的なプログラムの例として、簡単なゲームを作ってみる。最初に、画面にペンを作る:

```
カメ太＝タートル！作る。
```

次に、ペンを操作するボタンを作る。何をするボタンかわかるように、ここでは「前」という文字を書く。プログラムに次の 1 行を加えて実行すると、画面にボタンが作られる:

```
前進ボタン＝ボタン！”前”作る。
```

図 3.8: ボタンの作成

作ったボタンは、押してみても何もおきない。それは、押したときにすべき動作が定義されていないためである。そこで、ボタンを押すたびにペンが 10 歩ずつ進むようにする:

```
前進ボタン：動作＝「カメ太！１０ 歩く」。
```

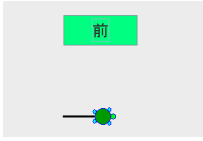


図 3.9: ボタンによるカメの操作

ここまでのプログラムはまとめると次の 3 行である:

```
カメ太＝タートル！作る。
前進ボタン＝ボタン！”前”作る。
前進ボタン：動作＝「カメ太！１０ 歩く」。
```

ここまでで、“カメ太” と “前進ボタン” という 2 つの「もの」を作り、前進ボタンには押されたときの動作を定義しした。この “動作” のように、「もの」には新しい命令を定義することができる。

ボタンを増やす

このままではボタンを押しても前進しかしないので、もうひとつボタンを作って回転できるようにしてみる。2 個目のボタンは、位置が重ならないように少し右にずらす。“x y 移動する” は、ペンを右に x 歩、上に y 歩動かす命令である。こちらのボタンを押したときには、ペンを右に 30 度回転させる:

```
右ボタン＝ボタン！”右”作る　１２０　０　移動する。
右ボタン：動作＝「カメ太！３０　右回り」。
```

図 3.10: 2 つのボタン表示

前進ボタンと右回転ボタンを使うと、いろいろな図形を描くことができる。実際に動作を確かめてみてほしい。

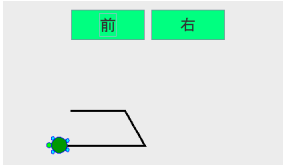


図 3.11: ボタンによるお絵かき

ここまでのプログラムをまとめて示しておく:

```
カメ太＝タートル！作る。
前進ボタン＝ボタン！”前”作る。
前進ボタン：動作＝「カメ太！１０　歩く」。
右ボタン＝ボタン！”右”作る　１２０　０　移動する。
右ボタン：動作＝「カメ太！３０　右回り」。
```

演習 ボタンでペンを動かすプログラムを打ち込んで動かせ。うまく行ったら、数値を変更してボタンを押した時に引ける線の長さや回転する角度が変化することを確認せよ。もっと多くのボタンを使って機能を色々選べるようにしてみるとなおよい。

3.2.2 アニメーション

左右ボタンを作る

先の例題をさらに拡張して行くことにする。先の例題では、回転が1方向だったので、2個のボタンを使って左右に向きを変えられるようにする。続いて、前進動作はボタンを押さなくても自動的に進むようにする。

まず、“前進ボタン” を左回転する “左ボタン” に変更する。プログラムの2、3行目は次のようだった:

```
前進ボタン＝ボタン！”前”作る。
前進ボタン：動作＝「カメ太！１０ 歩く」。
```

この2行を次ものに置き換える:

```
左ボタン＝ボタン！”左”作る。
左ボタン：動作＝「カメ太！３０ 左回り」。
```

これを実行すると、画面に「左」と「右」の2個のボタンが表示される (図 3.12)。

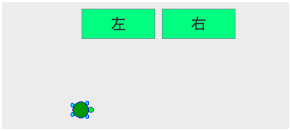


図 3.12: 左右ボタン

現在のプログラムを示す。2、3行目が「左ボタン」の定義に変更されている。

```
カメ太＝タートル！作る。
左ボタン＝ボタン！”左”作る。
```

```
左ボタン：動作＝「カメ太！３０ 左回り」。
右ボタン＝ボタン !” 右 ” 作る １２０ ０ 移動する。
右ボタン：動作＝「カメ太！３０ 右回り」。
```

ここで右ボタンだけ「移動する」という命令を使っているのは、移動しないと左ボタンと重なってしまい、左ボタンが隠されてしまうからである。そして、「120 0 移動する」というのは、横方向に 120、縦方向には 0 だけ移動するという意味である。

動作の自動実行

ドリトルのプログラムにおいて一定時間間隔で繰り返して動作を実行させるときは、「タイマー」を使う。ここでは、“時計” という名前のタイマーを作り、ある動作を 200 回実行するように設定する:

```
時計＝タイマー！作る ２００ 回数。
```

タイマーを動かすときには、実行したい動作を “「 」” に囲んで与える。ここでは、“カメ太！１０　歩く” というプログラムを繰り返し実行する。実行する時間間隔は、標準では 0.1 秒である。プログラムを実行すると、ペンが自動的に前進をはじめる。

```
時計！「カメ太！１０ 歩く」実行。
```

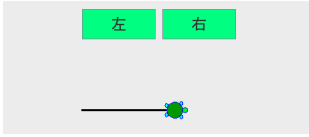


図 3.13: 自動的に動く

放っておくと、ペンはまっすぐに前進し、すぐに画面から外に出て行ってしまう。2 つのボタンでペンの向きを変えて、動きを操縦してみてほしい。

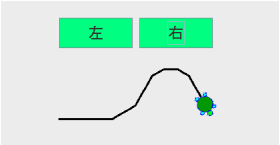


図 3.14: ボタンで操縦する

プログラムと手順的な自動処理

ここまでに見てきたプログラムはそれほど長いものではないが、これを動かすことで画面にさまざまなものを描くことができ、それなりに遊ぶことができる。

このように、「手順的な自動処理」によって作られた機構は人間が介在しなくても自動的に動くので、それと人間による制御を組み合わせることで「少ない制御で多くのことができる」ようになるわけである。このような用途に使えることも手順的な自動処理の 1 つの利点といえる。

実際にプログラムを作るときは、どのようなプログラムを作るかを決め、その計画に合わせてきちんと記述を組み立てて行く必要がある。このような「規則に従った書き方」はコンピュータに与えるデータ一般においても見られる性質だといえる。

全体として、プログラム制作を体験し、手順的な自動処理の性質を理解しておくことは、コンピュータを道具としてうまく活用していく上で有用だと言える。

演習 上の例題をさらに好きなように改良してみよ。どのような改良を行いたいかまず計画し、その計画に沿って行うこと。

3.3 コンピュータと手順的な自動処理

3.3.1 コンピュータの構造

コンピュータの構成要素

コンピュータの大まかな構造は 1 章でも取り上げたが、ここでもう少し詳しく見ておこう (図 3.15)。まず、コンピュータ本体の中はおおむね次のものから成っている:

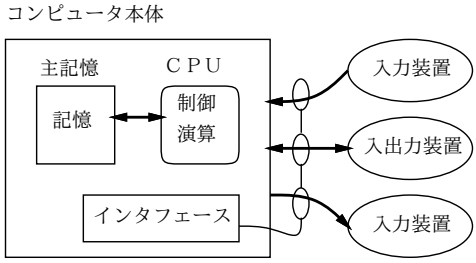


図 3.15: コンピュータの構造

- **CPU**(Central Processing Unit、中央処理装置) — コンピュータ本体の心臓部で、ここでプログラムを実行し、各種の処理を行う。さらに細かくは次のような部分に分かれている:
  - 演算回路 — 数値の演算など実際のデータ加工を行う部分。扱うデータを一時的に保管する少量の記憶装置も含まれている。
  - 制御回路 — プログラムに含まれる各命令の取り出し、解読、実行の制御を行う部分。
- **主記憶** (**メインメモリ**) — 実行中のプログラムやデータを保持する記憶装置。**アドレス** (番地) を指定して、そのアドレスにデータを格納したりそこに格納されているデータを取り出したりできる。
- **インタフェース** — 入出力装置と CPU のやりとりを仲介したり制御する部分。

コンピュータが外界と情報をやりとりするときは、すべて入出力装置を介しておこなう。これらには、次のようなものがある:

- 入力装置 — **キーボード**、**ポインティングデバイス** (位置入力および指示装置全般を意味し、**マウス**、**タッチパッド**、**タブレット**、**トラックボール**など多くの種類がある)、**ディジタルカメラ**や**スキャナー**、各種の**センサー** (温度、圧力、明るさ、等物理的情報を取り込む機器) などがある。
- 出力装置 — **ディスプレイ** (表示装置ないし表示画面)、**プリンタ**が代表的。また電気信号を通じてモータ、ヒータなど各種の機器を駆動することもある。

- 入出力装置 — 入力と出力を兼ねるもので。**サウンドボード**、**ビデオボード**などは音の信号の入力と出力、映像信号の入力と出力を両方行えるのが普通である。**ネットワークインタフェース**は情報の送信と受信両方に使われる。**補助記憶装置** (**ハードディスク装置**、**フロッピーディスク装置**、各種**外部メモリ**の読み書き装置) は情報の書き込みと読み出しの両方が行える。

ハードウェアとソフトウェア

前節で説明したような、コンピュータを構成する装置を一般に**ハードウェア**と呼ぶ。しかし、コンピュータはハードウェアだけでは動作せず、そこにプログラムやデータを加えてはじめて動作することができる。この、コンピュータの動作に必要なプログラムやデータなどを総称して**ソフトウェア**と呼ぶ。

コンピュータのハードウェアはさまざまな情報処理を行う機能を持っているが、具体的に**どのような処理を行うか**はそれ自体では決まらない。そしてこの「どのような処理を行うか」という**情報**を、ソフトウェアとして与えることではじめて具体的な処理を行うようになる。

このために、たとえば同じパーソナルコンピュータ上で、ワープロソフトを動かせば文書が作成でき、描画ソフトを動かせば絵を描くのに使うことができるわけである。この、ソフトウェアを取り替えることで (ハードウェアの能力内であれば) どのような処理でも行わせられることが、他の機器にないコンピュータの最大の特徴であり利点だといえる。

ソフトウェア自体はプログラムやデータの集まりであり、コンピュータ上で使用している間は他のデータと同様に主記憶に格納されている。したがって、プログラム自体を加工したり取り替えたりすることもコンピュータの機能によって自在におこなえる。この、プログラムもデータの一種として主記憶に格納する方式を**プログラム内蔵方式**と呼び、この方式が科学者フォン・ノイマンの提唱をきっかけに広まったことから、この方式によるコンピュータを**ノイマン型**コンピュータとも呼ぶ。

最初の実用となったコンピュータである **ENIAC** はプログラム内蔵方式ではなく、プログラムは多数の配線板によって表現されていた。このため、プログラムを取り替えるには配線板を抜き差しする必要があり、大変な手間が必要だった。

これに対し、今日のコンピュータはプログラム内蔵方式であり、常時動いている**オペレーティングシステム**と呼ばれるプログラムがハードディスクなどから必要なプログラムを取り出して来て主記憶に置くことで、必要なプログラムをすぐに動かすことができる。

なお、ここで言っている「CPU が実行するプログラム」は主記憶に格納されたビット列であり、先に学んだドリトル言語による記述のようなものとはだいぶ違っている。プログラミング言語による記述を CPU が実行できる形に直すのは、プログラミング言語処理系の役割である。

3.3.2　プログラムと手順的な自動処理

プログラムとその実行サイクル

主記憶上に置かれているプログラムは、CPU が実行する**命令**が並んだものであり、これを**機械語のプログラム**と呼ぶ。CPU は機械語のプログラム中の命令列をアドレスの順番に取り出して実行していく。命令の内容としては、主記憶とレジスタの間のデータ転送、レジスタにある値の演算や大小比較などがある。命令の中には次に実行する命令のアドレスを指定するもの (ジャンプ命令) があり、これによって命令列の一部を飛ばしたり同じ箇所を繰り返し実行したりもできる。

CPU の制御回路は次のような動作を繰り返し実行している。

- フェッチ — 実行すべき命令を主記憶から読み出す。
- デコード — 命令のビット列から実行すべき内容を解読し決定する。
- 実行 — 実際に命令の動作を実行する。より細かく見ると「命令で使うデータの読み出し」「動作の実行」「結果の書き込み」の 3 段階から成っている。

このような単純な動作を次々に極めて高速に (1 秒間に 1 億命令以上の割合で) 実行することによって、コンピュータは複雑な情報処理を実現している。このとき、一定時間にできるだけ多くの命令を実行するために、1 つの命令の処理が実行し終わってから次の命令の実行を開始するのではなく、命令を次々に重ね合わせながら実行していく。これを**パイプライン方式**という。さらに多くの命令を実行するため、順序関係に問題がない場合には同時に 2 つ以上の命令を実行する方式もある。

CPU 内部の動作は、**クロック**と呼ばれる周期的な信号に従って動いている。クロックの速度は、1 秒間あたり何回の信号周期があるかであらわす。これを**クロック周波数**という。たとえば 1GHz(ギガヘルツ) のクロックであれば、1 秒間におよそ 10^9 回の信号周期がある。

同じ種類の CPU であれば、一般にクロック周波数が高いものほど一定時間内に多くの命令が実行できると言える。CPU の種類が違えばこのことは必ずしもあてはまらない (CPU の機種ごとに、1 命令を平均何クロックで実行できるかも、1 つの命令で行える処理の量も違うため)。

手順的な自動処理

ここまでに説明したように、コンピュータの動作はソフトウェアによって決定される。そのソフトウエアの動作はプログラムという形で表現されている。プログラムは多数の命令の集まりであり、1 つひとつの命令が順番に実行されることで全体としての動作が進んで行く。

このような、あらかじめ決められた形で表現された手順にしたがって処理が進んで行く動作の形態を、ここでは**手順的な自動処理**と呼ぶことにする。手順的な自動処理には、次のような (これまでの生き物や機械にはなかった) 特性がある:

- 動作をあらかじめ決められた形で記述する
- 動作はその記述に厳密に従い、機械的に/自動的に進んで行く

たとえば、動作の途中で何らかの「判断」、つまり状況によって動作を変更することが必要であれば、その「判断する」という動作や、判断の結果がどうだったかに応じてそれぞれの場合に行うべき動作も、すべてあらかじめ決められた形で記述しておく必要がある。

これを人間に何かを頼む (命令する) ことと比べると、次のような大きな違いがある:

- 人間に頼む場合は「最終的にどうなって欲しいか (例:ある書類を 2 部コピーする)」を説明すれば、それを達成する方法は自分で判断してもらうことができる。しかし手順的な自動処理の場合は、すべての動作 (コピー機のところに行く、書類を読み取り装置にセットする、部数のボタンを押す、コピー開始ボタンを押す、等) を厳密に指定する必要がある。

- 人間に頼む場合は、細かいところは説明しておかなくても、適当に判断してやってもらえる。しかし手順的な自動処理では、どんな些細な動作でも指定されていない動作は実行されない。
- 人間の場合は、頼んでおいても忘れたり間違えたり勘違いしたりして、思った通りのことをやってもらえないことがあるし、途中で飽きたり疲れたりして仕事が終わらないこともある。しかし手順的な自動処理の場合はコンピュータによって実行されるので、ハードウェアの故障がない限り、指示した通りのことが寸分の違いもなく実行される (指示が間違っていれば、その間違った指示の通りのことが実行される)。

ではなぜ、「手順的な自動処理」が重要なのだろうか。コンピュータができる以前は、さまざまな判断や手順を必要とする作業は、たとえ機械的で決まり切った判断や手順であっても、人間がついていて実行したり判断したりするしか無かった。

しかし、「手順的な自動処理」が可能になると、あらかじめ手順を用意しておくだけで、あとは人間がついていなくても自動的に手順や判断を進めてもらうことができるし、コンピュータによる実行は高速で間違いもなく、多数の処理を行うことができる。これによって、「単純作業であるような頭脳労働」から人間を解放できるようになったことに、「手順的な自動処理」の価値があるといえる。

ただし上にも記したように、「手順的な自動処理」は人間に仕事を頼むのとは違った側面が多くある。そのため、その性質をよく理解してうまく使いこなすことが必要なわけである。

「手順的な自動処理」の構築

「手順的な自動処理」には上で挙げたようなこれまでにない特性があるので、それを組み立てる (構築する) ときにも「なんとなく」手順を組み立てるのではなく、それなりの方法が必要である。具体的には、次のような活動が必要になる:

(1) 問題を自らの判断に基づき定式化し、その解決方法を考える。
(2) 解決方法を、手順として組み上げ、自動処理可能な一定形式で記述した、コンピュータ上で実行可能なものとして実現する。

(3) 実現したものが問題解決として適切であるかを検証し、必要なら問題の定式化まで戻ってやり直す。

後で学ぶ問題解決プロセスの 1 形態だと考えることもできる。ただし、普通の問題解決であれば問題の解決を実践して結果を見るまでにはそれなりの時間がかかるので、何回もやり直すことはかなり大変である。これに対し、コンピュータ上で手順を組み立てて動かすことは自分ひとりでできることなので、納得するまでやり直して見ることができる。

「手順的な自動処理」の形態としては、たとえば表計算ソフトウェアのシート上で計算の手順を指定したり、物理シミュレーションソフトウェア上でシミュレーションする物体を配置したりすることなども含まれるが、プログラミング言語でプログラムを書くこともその 1 つといえる。

3.4 情報と問題解決

3.4.1 問題とその解決プロセス

問題とその性質

問題 (problem) とは一般に、解答ないし解決を要する事項や問いを意味する。学校の定期試験などに出される問いも「問題」であるし、我々が普段生活していて、解決を要する「問題」に出会うこともある。

たとえば、次のようなものはいずれも問題の例だといえる:

問題 1 $X^2 = 4$ を満たす X の値を求めよ。

問題 2 10 万円を年利何%で預けたら 10 年後に倍になるか知りたい。

問題 3 朝遅刻しない範囲で通学時間が最短の電車の選択を知りたい。

問題 4 修学旅行の自由行動で楽しめる訪問先を決めたい。

問題 5 朝の駅までの道で歩行喫煙者がいて煙いのを何とかしたい。

別の視点で言えば、「問題」とは「現在の状態と望まれる状態とに間に相違があること」であり、「解決策」とはその相違を解消する具体的な方法だと考えることができる。ここで「現在の状態」「望まれる状態」「解決策」(複数あるかも知れない) などは問題に付属する性質だといえる。

ほかに、「それが誰にとっての問題か」「解決されない場合のデメリットは何か」「解決できた場合のメリットは何か」などの性質もある。また、複数の解決策がある場合、それぞれのメリット、デメリットを検討して解決策を選ぶ必要があるかも知れない。

例: 数式を解く

たとえば、上の問題 1 について問題のさまざまな属性を挙げてみる:

- 問題 — $X^2 = 4$ を満たす X の値を求めよ。
- 現在の状態 — $X^2 = 4$ を満たす X の値が分からない状態。
- 望まれる状態 — $X^2 = 4$ を満たす X の値が分かった状態。
- 誰の問題か — 自分の問題。
- 解決できた場合のメリット — 気持ちがいい、宿題が出せる、試験の点数があがる、等。
- 解決できなかった場合のデメリット — 「解決できた場合のメリット」の裏返し。
- 解決策 1 — X にいろいろな数を入れて試す。
  - 解決策 1 のメリット — とりあえずすぐやってみられる。
  - 解決策 1 のデメリット — 解法として答案に書けない。もっと面倒な数値や一般式では適用できない。
- 解決策 2 — $X^2 - 4 = 0$ と 2 次方程式に変形して解の公式を使う。
  - 解決策 2 のメリット — 式のあてはめだけで済む。
  - 解決策 2 のデメリット — 大袈裟である。公式のあてはめが面倒で間違いやすい。解の公式を忘れていると使えない。
- 解決策 3 — 公式 $X = \pm\sqrt{\ } a$ にあてはめる。
  - 解決策 3 のメリット — 簡単な式のあてはめだけで済む。
  - 解決策 3 のデメリット — とくに思いつかない。

このように、解決策ごとの得失を整理して検討することで、やみくもに行動するよりもうまく問題に対処できる。

問題解決プロセスとその必要性

前節では問題の解決策がいきなり出てきたように見えたが、多くの問題ではそう簡単に解決策が見つかるとは限らないし、解決策を検討する前にさまざまな情報収集が必要になることもある。またそれ以前に、解くべき問題がどのような問題なのか明確でない場合もある。さらに、解決策が見つかったと思っても実際にやってみるとそれでは十分でなかったということもあるし、その場合は別の解決策を見つけてやり直すことが必要になるだろう。

これらのことがらを考えると、問題解決には「問題をはっきりさせる」「情報を収集したり分析する」「解決策を考える」「解決策が有効かどうか調べる」「有効でなければやり直す」など多くの段階が含まれている。

一般に、あることを達成するためのさまざまな作業を一連のつながりとして捉えたものを**プロセス** (過程) と呼ぶ。問題解決の場合、そのプロセスは一般に次のような形を取ると考えられる。

(1) 問題の認識/同定/記述 — 問題の存在を認識し、それがどのような問題なのかをはっきりさせる。
(2) 問題の分析 — 問題がどのような構造を持っているのかを検討し、また問題や解決策の検討に必要な情報を集める。
(3) 解法の設計/実現 — 解決策の構成やみちすじを考えて決め、実際にその解を実現してみる。複数の解法から 1 つ選択することもこの段階に含まれる。
(4) 検証/反復 — 解を実現したものが実際に問題を解決しているかどうかチェックし、不十分な点があれば (1) や (2) や (3) に戻ってやり直す。

非常に簡単で分かりやすい問題の場合、このプロセスは瞬時に頭の中で行われてしまい、はっきりしないこともある。しかし、込み入った問題の場合は、これらの段階をそれぞれ意識して、考えたことを逐一記録しながら問題解決を進めて行くことが、考えおとしや勘違いを防ぎ、有効な解を導くことにつながる。

3.4.2　問題解決プロセスの例: 複利計算

認識/同定/記述

ここでは問題解決プロセスの具体例として前述の問題 2(複利計算) を取り上げ、そのプロセスを見て行くことにする。まず問題の属性で分かっていることから記述してみる:

- 問題の認識 — テキスト (本書) に例題として挙げられていた。
- 問題の同定 — どこの金融機関のどの商品、といった調査は行わず、「金利がどれだけだったら」という単なる計算上の問題として扱う。
- 問題の記述 — 10 万円を年利何%で預けたら 10 年後に倍になるか知りたい。

この例では「問題の同定」の内容が以後の問題解決プロセスに大きな影響をもたらすことに注意したい。たとえば、「実際に 10 万円預けて 10 年で倍にしたい」という立場で考えた場合は、現在の金利情勢では通常の金融商品でそのような高い金利のものはないので、株や海外金融商品などの商品を探求することになり、かなり違った枠組の問題になってしまう。ここではこの問題をそのような立場ではなく、「仮に金利何%だったら」という計算上の問題として扱うこととする。

問題の分析

問題の分析として、この問題では「年利何%」の意味を明確化することが重要である。ここでは、「年利 r%」で「現在の手持ち金額」が「M 円」であれば、「1 年後の手持ち金額 M' 円」は次の式で表せるはずである (1 項目が元金、2 項目が利息):

$$M' = M + M \times \frac{r}{100}$$

ここで式をできるだけ簡単にするため、定数 C を $C = (1 + \frac{r}{100})$ と置くと、次のようになる。

$$M' = C \times M$$

たとえば、年利 1%なら $C = 1.01$、2%なら $C = 1.02$ である。このように C を置くと、10 年後の手持ち金額は次の式で表せる:

$$M_{final} = C \times C \times C \times C \times C \times C \times C \times C \times C \times C \times M$$

最終金額が「元金の 2 倍以上」ならいいのだから、問題は次の式を満たすような C(ひいては r) を求めることになる:

$$C \times C \times C \times C \times C \times C \times C \times C \times C \times C \times M \geq 2 \times M$$

こうして見ると、M は何でもいい (もちろん、10 万円が 2 倍になる金利なら、100 万円でも 2 倍になるに決まっている):

$$C \times C \times C \times C \times C \times C \times C \times C \times C \times C \geq 2$$

このように、分析を進めていくと問題の内容が整理され (ある意味ではモデル化され)、「何らかの方法で上記の式を満たす C や r を求める」という形に分かりやすくなったことが分かる。また、この分析には具体的な金融商品や金融機関などは出てこないが、それは先の問題の同定/記述からは当然のことなわけである。

解の設計/実現

問題の分析によって問題が整理できたので、解を求める方法が複数思い付く状態となった:

- 解法 1 — 「2 の 9 乗根」を計算できる関数電卓で計算する。
- 解法 2 — r として 1%、2%、…を与えて実際に $C^9 \geq 2$ になるような最初の r を見つけるプログラムを作成する。
- 解法 3 — 表計算ソフトで r として 1%、2%、…についての計算をおこない、「2 倍になる」ところは人間が目で見て見つける。

どの方法もそれぞれ優劣があるが、ここでは「計算過程が見えて納得しやすい」ことから解法 3 を選択してみる。そこで、次にワークシートの設計を次のように行う:

- A 列には見やすさのため年数を 1、2、…10 まで入れる (セル A2 を 1 とする)。
- B〜K 列の 1 行目には利率として 1%、2%、…10%までのときの C を入れる。
- 2 行目は 1 年目に対応させるため「1.00」を入れる。

I11 =I10*I$1

	A	B	C	D	E	F	G	H	I	J	K
1		1.01	1.02	1.03	1.04	1.05	1.06	1.07	1.08	1.09	1.10
2	1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
3	2	1.01	1.02	1.03	1.04	1.05	1.06	1.07	1.08	1.09	1.10
4	3	1.02	1.04	1.06	1.08	1.10	1.12	1.14	1.17	1.19	1.21
5	4	1.03	1.06	1.09	1.12	1.16	1.19	1.23	1.26	1.30	1.33
6	5	1.04	1.08	1.13	1.17	1.22	1.26	1.31	1.36	1.41	1.46
7	6	1.05	1.10	1.16	1.22	1.28	1.34	1.40	1.47	1.54	1.61
8	7	1.06	1.13	1.19	1.27	1.34	1.42	1.50	1.59	1.68	1.77
9	8	1.07	1.15	1.23	1.32	1.41	1.50	1.61	1.71	1.83	1.95
10	9	1.08	1.17	1.27	1.37	1.48	1.59	1.72	1.85	1.99	2.14
11	10	1.09	1.20	1.30	1.42	1.55	1.69	1.84	2.00	2.17	2.36

図 3.16: 表計算ソフトによる計算

- 3 行目〜11 行目には「その年の結果」を「1 つ前の年に C を掛けた値」として求める。それは、たとえば B 列なら「`B3 = B2*B$1`」の計算式で与えられる。[2]

これを表計算ソフトによって計算しているようすを図 3.16 に示す。見やすさのため書式は小数点以下 2 桁にした。これから見ると、年利 8%あれば 10 年で倍になることが分かる。

検証/反復

この問題では検証といってもとくに問題はないように思えるが、実際に「$10,000 \times 1.08^9$」を電卓などで計算してみる (または、表計算ソフトでも 10 万円を入れてみる) と、結果は「199,900.46 円」で 100 円ほど目標に足りないことわかる (表計算ソフトの書式を小数点以下 2 桁に変更しないでおいても、このことは分かる)。こうしてみると、誤差についてどのような立場を取るかについて検討していなかったことが分かる。

このほか、作成したワークシートを自分だけが使うのか、他人にも使わせるのか (それならもっと見出しつけた方がよかった) など、動かしてみてから分かる疑問点はいろいろあるはずである。これらについて再度検討し、必要なら前の段階まで戻ってやり直すことで、問題解決プロセスが進んで行くことになる。

[2] `$`がついているのは、これがついていないと式を B4 にコピーした時に「`B4 = B3*B2`」のように調整されてしまうのを防ぐため。

より広範囲な問題

ここで例として取り上げた問題は定式化が比較的簡単な部類のものであったが、より複雑な問題の場合は、さまざまなモデルを検討する必要があったり、モデルの解法もすぐには見つからなかったりする。また、社会的なことがらを対象とする問題では、モデルを作って解を検討しても、実際の社会がその通りには動かないことが多い。このように、問題解決とは非常に奥行きの深い分野だといえる。

第4章　情報社会

4.1　現代社会と情報技術

4.1.1　身の回りの情報機器とコンピュータ

現代社会は情報社会であり、我々の身の周りには多くの情報機器が存在している。また、もともと情報機器ではないのに、コンピュータを内蔵するようになったものも多数ある。ここではそれらについて整理してみる:

テレビ/ラジオ/電話/FAX これらはコンピュータ以前からある情報伝達手段のための機器であり、今でも変わらず存在している。しかし、以前はこれらでしか手に入らなかった情報がコンピュータとネットワークを通じても入手できるようになったため、これらの重要性は低下していると見る向きもある。また、これらの機器の内部にも今日では小型のコンピュータが内蔵されてさまざまな処理を行うようになっている。

携帯電話 最近では多くの人が携帯電話を持ち歩いている。最近の携帯電話の中にはコンピュータが入っている。コンピュータが携帯電話に備わっていることにより、電子メールの読み書や WWW の閲覧が可能になった。電子メールを読み書きするとき「かな漢字変換」や「文書の編集」を行うが、この機能を実現するためにもコンピュータの存在が不可欠である。この他、最近の携帯電話はゲームで遊べたり、財布の代わりになったりすることができる。これは携帯電話の中心部分がコンピュータであり、コンピュータプログラムを追加することが比較的簡単にできるからこそ実現していることである。

パソコン 最近の学校や会社ではパソコンが多く使われている。パソコンを持つ家庭も増えている。パソコンはインターネットに接続さ

れる場合が多い。パソコンには汎用的な基本ソフトウェア (OS) が搭載されており、ソフトウェアを追加することにより、利用者自身によって、利用者が要求する機能を持たせることが可能な機械である。携帯電話と同様に、パソコンで電子メールの読み書きや WWW の閲覧ができる。WWW を使ったインターネットショッピングも多く使われるようになった。家庭・学校・職場で何か文書を作成するとき、パソコンのワープロソフトを使う場合が多くなった。ワープロソフトにより、気楽に活字体の文書を作成することができるようになった。また、間違いの訂正など文書の編集が容易になった。表計算ソフトにより、様々な身近な場面で必要になる会計処理が容易になった。表計算ソフトも、数値の訂正などが容易にできる。デジタルカメラやデジタルビデオで撮影した静止画や音声や動画を表示したり、編集したりする機能を持っているソフトウェアも販売されている。これらのソフトウェアとインターネットを使って、個人が作成したマルチメディア作品を世界に向けて発信することも容易になった。

家庭用電気製品 最近の電子炊飯器にはコンピュータが入っていることが多い。温度センサーとヒータなどをこのコンピュータに接続し、おいしいご飯が炊けるように制御している。コンピュータが使われる以前は、サーモスタットと呼ばれる、温度を一定に保つための簡単な自動温度制御装置が使われていたが、サーモスタットでは細かな温度制御を行うことがむずかしかった。コンピュータによりそれが簡単になった。また、コンピュータが搭載されることにより、タイマーなどの様々な付加機能が搭載できるようになった。近年、家庭電化製品の多くにコンピュータが使われるようになっている。

自動車 最近の自動車には、エンジンの制御、スピードや燃料残量の表示、カーナビ、オーディオ、エアコン、パワーウィンドウ、アンチロックブレーキシステムオートマティックトランスミッションなどの機能それぞれに対してコンピュータが使われている。自動車は、コンピュータが普及する以前から存在していたが、近年、それまで使われていた制御用の機械部品や電気部品の代わりにコンピュータが使われるようになった。

情報機器の目的はもともと、情報を扱うことであるが、従来からの情報機器ではコンピュータの発達によって人間がいちいち指図しなくても多くの動作を適切に行ってくれるようになっている。また携帯電話やパソコンの場合は、ディジタルデータであれば何でも送受信でき扱えるというコンピュータの特性を活かして、多様な情報を柔軟に扱えるようになっている。

情報機器以外の機器や装置でも、コンピュータを組み込むことによってこれまでにない細かい制御を行わせたりさまざまな状況に柔軟に対処できるようになっている。またホームネットワークなど、これまでは独立していた装置どうしで情報を有機的に交換することでより役立つ機能を提供しようとする試みもなされてきている。

課題 なぜ、制御用の機械部品や電気部品の代わりにコンピュータが使われるようになったのだろうか? 理由を調べてみよう。

4.1.2 さまざまな情報システム

情報システムとは、情報を扱うことを目的として多くの構成要素を組み合わせて構築されたもの (システム) をいう。我々の社会には、直接個人が接することがないものも含め、多種類の情報システムが存在し稼働しており、これらの活動が社会を支えている。ここでは具体的な情報システムの例をいくつか見てみる:

銀行オンラインシステム 銀行では膨大な量の計算が日夜行われている。個々の預金者に対して通帳の預金、預金引き出し、利息などの計算を間違いなく行わなければならない。従って多くの銀行は早くからコンピュータシステムを導入していた。その後、窓口業務の負担を軽減し、預金者を確保するため、ATM 端末が使われるようになった。ATM 端末は銀行の中央のコンピュータと通信回線で結ばれ、入金や出金が ATM 端末がある場所ですぐに行えるようになった。銀行とその銀行と取引がある会社が通信回線で結ばれ、給与の自動振込みが行われるようになった。また、銀行のコンピュータ同士も通信回線で結ばれ、預金者が別の銀行の預金者に振込みを行ったり、異なる銀行の ATM 端末でも入金や出金が行えるようになった。銀行オンラインシステムのネットワークは

海外の銀行やクレジットカード会社ともつながり、いまや世界中に広がった大規模コンピュータネットワークとなっている。

鉄道・航空機予約システム 日本では 1960 年に旧国鉄が世界で最初の鉄道オンライン予約システム MARS の運用を開始した。コンピュータを使うことにより、人間の手作業ではさばききれなかった数多くの予約処理が可能になった。現在では鉄道や航空機などの公共交通機関の座席予約は WWW 上の予約システムを使って個人が自宅や携帯電話により行うことが可能になった。コンピュータとネットワークの普及なしには、このような予約システムの実現は困難である。(参考:情報処理学会編「鉄道とコンピュータ」情報フロンティアシリーズ、共立出版,1998)

交通運行管理システム 鉄道では信号の扱いやポイントの切り替えに間違いがあると、まず確実に大事故が発生してしまう。航空機でも航路の間違いや機器の扱いの間違いは機体を危険にさらすことになる。過去においては、このような管理をすべて人間が行っていたが、人間はどんなに注意しても間違いを侵すことは避けられず、これに起因する事故が多く起こってきた。現在では、このような管理にコンピュータが使われるようになっている。コンピュータには「うっかり」「不注意」はないので、過去のような不注意に基づく事故は劇的に減らすことができた。また、このような「安全のための」システムに加えて現在では、できるだけ効率的に車両や機体を運用したり、事故や故障などでスケジュールが乱れたときに速やかに復旧させたり、乗客に情報を提供するなど、「効率」「便利さ」のためにも情報システムが使われるようになってきている。(参考:情報処理学会編「鉄道とコンピュータ」情報フロンティアシリーズ、共立出版,1998)

工場における生産管理システム 工場で何かを作るとき、その工程を細かく分けて分業することが行われる。このとき、それぞれの工程に対して、必要な部品が必要な時間に無駄なく供給される必要がある。このような生産管理を行うためにコンピュータが使われる。また、単純作業そのものを人間の代わりにコンピュータに制御されたロボットが行うことも行われる。

POS/販売管理システム スーパーマーケットやコンビニエンスストア

では、レジで商品が販売されるごとにそのことがセンターに連絡され (POS — Point Of Sales)、その情報を用いて売れた商品を迅速に補給したり、さらにデータを分析して「売れ筋」の商品を充実させ売れ行きを増大させるなどの活動が行われている。その他の小売業や小売以外の販売業種でも、商品の受注や発送などはコンピュータによって管理され、いちいち手作業で集計しなくても会計処理が行えるとともに、データが集積されて分析が行えるようになっている。

さまざまな情報システムはもともと、人間が行っていた会計や注文などの情報処理を効率化するために作られ始めたが、今日では単なる効率化にとどまらず、これまでになかった新しい価値やサービスを提供することにも使われるようになってきている。その具体例としては、POS の項で挙げたようなデータの分析活用や、鉄道の予約システムでオペラのチケットが買えたりコンビニで電気代や携帯料金が払えたりなどのサービスの多様化が代表的である。

現在、新聞社のほとんどはコンピュータを使って新聞の編集を行い、直接印刷機で紙面を印刷している。それ以前は活字を使った活版印刷が行われていた。最初に開発されたコンピュータを使った新聞発行システムは NASA のアポロ計画に匹敵する大規模プロジェクトと言われていた」(参考: 杉山隆男「メディアの興亡」)

課題 ここで挙げた以外の種類の情報システムがないかどうか、考えてみよう。また、そのようなシステムについて検索し、実際にどのようなものでどのような性質があるか調べよう。

4.2 さまざまなストーリー

4.2.1 ネット社会と個人

演習 情報社会に関わる、以下の架空の事例について、どのような問題があるか、自分ならどう行動するかを考えてみよう (事例部分を読んで考えをメモし、解説を読んだ後で再度検討する。また友人とメモを見せ合って考え方の違うところを検討するのもよい)。

電子メールの Bcc には気をつけよう

竜太くんはある日、友人の淳一から陽子さんあてのメールの写しを受け取った。それは次のようなものだった:

```
From:  jun-ichi@exmample.net
To:  yoko@example.net
Date:  Fri Sep 15 11:15:04 JST 2006
Subject:  ハイキング

陽子さん:
昨日、ハイキング行きたいと言っていたよね。
僕の方で男子 3 人くらい声掛けてみるからそっちも
女子あと 3 人くらい集めて企画しない?
                                        淳一
```

そこで竜太君はさっそく「それ賛成!」と返事のメールを出したのだが、翌日淳一に「まだ交渉中なのにいきなり出てこないでよ!!!」と言われて「???」になってしまった。

メールヘッダにはそのメールが誰あてかの情報が含まれている。まさと君のアドレスがヘッダに含まれていないことに注意。これはおそらく、淳一くんが Bcc:(Blind Carbon Copy、内緒の写し) の機能を使って、陽子さんには内緒でそっと竜太くんに写しを送っていたものと判断できる。それなのに、竜太くんは返事を出すことでそのことをバラしてしまったわけである。

竜太くんがヘッダに注意して、自分が内緒の写しを受け取っていると認識しておけばトラブルは避けられた。しかし淳一くんも Bcc:にはそのような問題があると認識して使用を避けた方がよかったかも知れない (メッセージの写しをコピーして普通のメールで竜太くんに送ることはできたはずである)。ところでそもそも、誰かあてのメールの内容を別の人こっそり知らせることはどうなのだろう? [1]

[1] 日常生活でも誰かと会話した内容を別の人に知らせることはあるはずである。ということは、一律によくないというような問題ではなく、その場ごとの判断だと言えるだろう。

メールではきちんと名乗ろう

建太くんはこれまで携帯メールしか使ったことがなかったが、自宅でプロバイダを変更したのに伴って自分のメールアカウントを貰うことができた。そこでさっそく、そのアカウントから友人の淳一のアドレスにメールを出してみた:

```
From:  bxq0234@server01.baxsv.net
To:  jun-ichi@exmample.net
Date:  Fri Sep 22 13:22:04 JST 2006

明日朝、駅で 8 時に待っててくれる?
```

翌朝 8 時に駅でしばらく待っていたが淳一は来なかった。学校で淳一に「なんで連絡したのに駅で待ってないんだ」と文句を行ったところ「え? そんな約束知らないよ?」と言われてしまった。

これは淳一が気の毒で、誰とも分からない人から「8 時に待っていろ」というメールが来ても何のことだか分からないだろう。または、知らない人からのメールを読まないように設定していたかも知れない。

携帯メールは文面を短くしがちだが、通常のメールではきちんと自分が誰だか名乗り、Subject: などもつけて形を整え、丁寧に用件を述べるべきである。それでも知らないアドレスのメールを読まない設定だと効果がないが、その場合は知合いに「新しいアドレスとしてこれこれを使い始めました」という連絡を別途する必要があるかも知れない。

チェーンメール

竜太くんの叔父は病気で入院していて、手術のためにある特殊な血液型の献血を必要としていた。献血が不足していると聞いた竜太くんは自分の出入りしているあちこちの掲示板に次のようなメッセージを書き込んだ:

> 僕の叔父の手術のために必要な、○×型の血液の献血が
> 不足しています。献血できる人は病院 (03-xxxx-xxxx) に
> 連絡してください。またこのメッセージをできるだけ
> 多くの知合いに転送して広めてください。

数日後、病院にお見舞に行った竜太くんは「病院の電話がパンクして大変だった事件」のことを聞いて「???」になってしまった。

メールに限らず、メッセージの増殖を呼びかけるものはすべて「チェーンメール」と同じ性質があり、同じ問題がある。すなわち、大量のメッセージトラフィックによるネットワークの負荷となり、またそこに言及されているサイト等にアクセスが集中して使用不能になることがある。竜太くんの場合は善意でやったことではあるが、結果的にはチェーンメールと同じ迷惑を作り出してしまったわけである。

コミュニティごとの常識

正人くんはいくつかの掲示板で趣味の切手についてやり取りを楽しんでいる。ある日、最近始めたらしい切手について語っているブログを見つけて読んでいたら、ある切手の発行年について勘違いしているようだったので、そのブログの掲示板に行っていつもの調子で「その○○の発行年は 1987 年でしょ。」とひとこと書き込んだ。翌日どうなったかなと思ってその掲示板を見てみたら「誰これ? 自己紹介もしないで頭おかしいんじゃない?」「空気嫁だよな」などの書き込みが多数あって「???」になってしまった。

コミュニティにはコミュニティごとの慣習や常識がある。このブログの掲示板では、始めての人は名乗って自己紹介をするなどの「仁義」が常識だったらしいのに、正人はこれまで自分が使っていた掲示板と同様のつもりで書き込んでしまい、そのコミュニティの「常識」に外れた行動をしてしまったわけである。このような無用のトラブルを招かないためには、掲示板の以前の書き込みを少し読むなどして、様子を知ってから参加し、なおかつ様子が分かるまでは丁寧めにふるまうなどの配慮が望まれる。

4.2.2 ネット社会の決まりと個人情報

テレビ番組の録画

竜太くんは先週の日曜日、帰宅が遅くなって楽しみにしていた連続ドラマを見逃してしまい、悔しい思いをしていた。昨日そのドラマのファンサイトの掲示板でそのことを書き込んだところ、実費で録画したテープのコピーを送ってあげるという親切な人が現れた。しかし今日、陽子さんと雑談していてそのことが出たら、陽子さんは「それって著作権法違反だよね」と言う。本当だろうか。

テレビの番組は、テレビ局が著作権を持つ著作物であり (さらに個別の番組ごとの著作権、使われている音楽に関する著作権など多数のものが関連している)、それを無断で複写することは著作権の侵害になる。だからコピーを頼むべきではない。

普段、テレビ番組を録画して後で見たり、レンタルした CD をコピーして繰り返し聴いたりできるのは、あくまでも「個人の私的複製」だからである。私的複製が認められるのは家族などごく近い間柄に限られることに注意。

引用の条件

竜太くんは自分のサイトを魅力的なものにしたいと思い、そのトップページに好きなバンドの歌の歌詞を入れてデザインし、公開開始した。しかししばらくしたら、それを見た複数の人から「歌詞は著作物だから許諾なしに勝手に公開してはいけないだろう」という指摘が届くようになった。先生には「引用は許諾なしに行っても構わない」と教わったので無視するつもりなのだが…

確かに引用であれば許諾は不要だが、それには引用の条件、すなわち (1) 出展を明示し、(2) 必要最小限の範囲で、(3) 引用される内容が従となる形であることが必要となる。たとえばその歌詞について論評するため必要なところだけを挙げるのなら構わなかっただろうが、自分のページを魅力的にするために、歌詞をそのまま載せるのでは、引用とは言えないだろう。

ちなみに、写真などの画像は「一部だけを引用」(つまりトリミングして必要部分を取り出す) すると、無断での内容改変 (これも著作権法で禁止されている) になってしまうので、実質的に引用はあり得ない。

送信可能化権

洋子さんは美術部で絵を描いている。写真部の夏実さんが洋子さんの作品の 1 つを気に入ってデジカメ写真に撮りたいと言ってきたので「他人には渡さないでね」と断って撮影させた。翌週、洋子さんが公開している Web ページを見たらその絵の写真が掲載されていたので文句を言いにいったが「私はサーバのハードディスクに画像データを置いただけだから他人には渡していないよ」と言い返されて「???」になってしまった。

過去に夏実さんと同じ言い逃れをする人が実際にいたため、著作権法が改正され、「他人が取得できる形でサーバなどに置く」ことの権利を**送信可能化権** (公衆送信権の一部にあたる) として明記するようになった。したがって、夏実さんが著作者である洋子さんの許諾なしに画像データを送信可能にしたことは著作権法違反に当たる。

個人情報の保護

竜太くんはブラズバンド斑の総務担当をしている。今年も合宿の季節になったので、合宿参加者の名簿を作成し、コンビニのコインコピーで人数ぶんコピーした。そのとき原稿をコピー機に忘れて来てしまったが、コピーは多めに取ったので問題ないと思っていた。ところが、夏休みが終わった頃、合宿の参加者にだけ、楽器の通販などのダイレクトメールが多数届くようになってしまった。何が起きたのだろう?

名簿の情報というのは商品価値があってそれなりの値段で売買されている。誰かがコンビニのゴミ箱などから原稿を拾って、小遣い稼ぎに名簿業者に売ったのかも知れない。

個人の情報が載ったものは注意深く取り扱うべきであった。今からでも、ダイレクトメールの送付元に問い合わせて名簿の由来を調べ、使用をやめてくれるように頼むことはできるかも知れない (大変だけれど)。

ポイントカードと個人情報

洋子さんはあるスーパーで半月前に購入した下着がよかったのでまた買おうと思って行き、商品棚に見当たらないのでカウンターの店員に相談した。すると店員は「ポイントカードをお持ちですか」と言い、洋子さんがカードを出すと何やらパソコンを操作していたが、まもなく「9 月○日に○○社の××を購入されていますね」と言った。洋子さんは驚いて「そんなことまで記録されているのですか。プライバシー情報じゃないのですか…」と言ったが、店員の返事は「ポイントカードの規約に『当社が業務上必要な範囲で会員に関する情報を利用』とあり、加入時にご承諾頂いたはずですが」だった。

小売業が発行するポイントカードの目的は、割り引きサービスなどで客を囲い込むこともちろんあるが、レジで清算時に POS 情報とポイントカード番号を一緒に記録することで「誰が何をいつどれだけ買ったか」を記録して品揃えや購買行動の分析に役立てることもある。

規約の文面からだけではそこまで想像できないだろうが、そのことは理解しておいて、記録されたくない場合は購入時にポイントカードを提示しないなどの自衛策を取るようにするのがよい (これらの情報が間違って漏洩してしまうなどの事件もあり得ないことではない)。

4.2.3 情報社会の安全性

ウィルス

邦子さんはある日、進一から「このゲームプログラム面白いよ」と書かれたメールを受け取った。メールには実行形式のプログラムが添付されていたので、さっそく開いて実行してみたが「うまく動作しません」というメッセージが出るだけだった。しかし翌日パソコンを起動しようとしたら立ち上がらない。学校で進一に事情を聞こうとしたが「そんなメール送っていないよ」という返事だった。

ウィルスの中には感染するとアドレス帳にある名前に片端から自分を添付したメールを送信するものがある。そのとき、差出人も偽造される。邦子さんが受け取ったのもそんなメールであり、差出人が友人だったためつい開いて感染してしまったと思われる。

対策としては、パソコンを縮小モード (セーフモードなどとも呼ぶ) で立ち上げてみて、立ち上がればウィルス駆除ソフトで駆除を試みる。それが駄目なら、OS から再インストールすることになる。ともかく、素性の分からないプログラムの実行は絶対に避けるべきであった。

フィッシング

永瀬氏は A カードというクレジットカードの会員である。ある日、カード会社から「Web でカードの更新手続きをしないと期限切れになります」というメールが来た。そのメールに記載された URL を開いたら、確かにカード会社のロゴがあるサイトが表示され、SSL による暗号通信になっていて、ブラウザの警告も出なかったので、そこでカード情報を打ち込んで手続きを済ませた。ところが翌月、カードの請求書を見たら身に覚えのない高額な宝石の買物をしたことになっていた。何が悪かったのだろう?

永瀬氏はニセのサイトに誘導して重要な情報を打ち込ませて盗むという「フィッシング詐欺」に引っかかったと思われる。Web サイトの内容は作成者がどうにでも作れるので、会社のロゴがあっても安心はできない。カード会社が Web 上でカード情報を打ち込ませることは無いはずだから無視すればよかったのだが、もしどうしても心配なら電話などの手段でカード会社に確認するべきだった。ブラウザに表示されている URL を確認するという方法もあるが、URL は偽装されてしまうこともあるので万全の方法とはいえない。

SSL による暗号化通信は、CA(認証局) にお金を払えばどのような組織でも適正な証明書が得られるので、ブラウザの警告が出なかったというだけでそのカード会社だとは確認できない。もう少し用心して、SSL 接続のページに入ったところでブラウザで「証明書情報」を表示させて確認すれば、その証明書がカード会社のものでないことが分かったはずである。

バックアップ

進一くんはパソコンを使って夏休みの課題を作成していたが、近くに雷が落ちていきなり停電し、パソコンの電源も落ちてしまった。停電が回復したので続きをやろうと思ってパソコンの電源を投入したが、立ち上がらない…何日も掛かって仕上げた課題なのに (号泣)。

雷が近づいている時にパソコンでの作業はやめておくべきだったし、サージプロテクタや無停電電源などがあれば悲劇は回避できたかも知れないが、何よりも重要なデータは定期的に別の場所にバックアップして保管しておくべきだった。ハードウェアはどんなものでもいつか必ず壊れる。だから、バックアップは常に最悪の場合を考えて怠らないようにするべきであった。

システム障害

洋子さんは念願の大学に合格し、これから合格手続きに行くところだった。入学金を今日中に入金しないと入学資格が取り消しになってしまうのだ。ところが銀行の ATM の前に行くと人が沢山行列して騒いでいる。システム障害があって ATM がうまく動作せず、一人あたり非常に時間が掛かっているらしい…このままでは時間に間に合わないかも!?

このような場合であれば、大学に連絡すれば事情を考慮してくれる可能性が高い。しかし、このような場合に備えて、利用する銀行を複数に分散しておくなどのリスク対策も必要だったかも知れない。場合によっては本当にお金が出せないとアウトになるような事態もあるわけだから…

4.3 情報社会における変化と問題点

4.3.1 情報社会における変化

コンピュータとネットワークの影響

情報処理装置としての人間そのものは、太古の昔から今日に至るまで変わっていないが、コンピュータという「情報処理の道具」の登場により、我々が情報を取り扱う能力は飛躍的に増大した。具体的に変化した点を挙げてみよう:

- コンピュータの発達により、大量の情報を (人間の) 外部に保管し、この中で必要なものを短時間で検索し、取り出すことができるようになった。
- コンピュータの発達により、大量の情報を人間が行うより遥かに早く正確に、加工・計算することができるようになった。
- コンピュータの発達により、人間があまり得意でない、単調な仕事の繰り返しをコンピュータと周辺機器に肩代わりさせることができるようになった。
- コンピュータにつながったネットワークの普及と発達により、情報の到達速度が圧倒的に早くなり、情報到達範囲が圧倒的に広くなった。これに従って、より多くの人が、必要な情報を、より早く入手することができるようになった。
- ネットワークの普及と発達により、より多くの人が、手軽に情報発信を行うことができるようになった。

情報社会の光と影

今日の情報社会では、コンピュータ、ネットワーク、情報機器、情報システムの普及により、我々はより多くの情報を効率よくやり取りしたり、短時間/低コストでさまざまな社会活動に関与できるようになってきている。

しかし、どのような事柄であっても、プラスの面とマイナスの面が相伴うのが普通であり、これは上に挙げたことがらについても変わるところはない。ここでは、前節までに延べて来た情報技術や情報システムの発達と普及がもたらすマイナス面についても考えてみる。

ここで重要なのは、これらのマイナス面に対する意識の持ち方である。たとえば「新しいものだからマイナス面があってもしかたない」と目をつぶっていると痛い目に合う可能性が高いし、逆に「マイナス面があるからこのようなものは使わない」という短絡的な考えをすると、せっかくの技術進歩の恩恵を受ける機会を損なうことになる。

そうではなくて、マイナス面についてもしっかり理解した上で、その影響をできるだけ受けないためにはどのように行動すべきかを各自が考える必要がある。そして、さまざまな技術やシステムごとに、そこから受ける恩恵と被害にあった場合の損失を考慮した上で、使うかどうかを判断する姿勢を持つことが望ましい。

以下では情報社会におけるさまざま問題について分類して見て行くことにする。

4.3.2 情報社会の問題点

サイバー犯罪

サイバー犯罪とは、コンピュータやネットワークの機能を駆使することではじめて可能になるような種類の犯罪全般を指す。過去においては「インターネット上には法律は及ばない」というとんでもない主張を真面目に延べる人もいたが、ネットワークもコンピュータも現実社会の一部であり、日常生活の他の場面と同様に法律やマナーが適用されることをよく理解しておくべきである。

では、日常生活と同じなら、なぜサイバー犯罪がことさらに問題になるのだろうか。それは、サイバー犯罪の踏み台となっているさまざまな情報技術が新しいもので、そこにどのような問題や危険性があるかを我々が十分理解していない面があり、犯罪者がそこにつけこんで来やすいという点があるからである。

したがって、ひとくちにサイバー犯罪に注意するというだけでなく、具体的にどのような問題が起きているかについて知っておき注意することも必要である。以下でサイバー犯罪の代表的なものについて簡単に説明しておく:

- コンピュータウィルスの感染 — **コンピュータウィルス**とは、広義には自己増殖機能を持つプログラムを言う。多くのウィルスは悪意をもつ人物によって製作/流布され(これは犯罪行為)、感染するとコンピュータの動作が不安定になったりファイルの消去など有害な動作を行う。ウィルスへの感染をを避けるには、正体の分からないプログラムを実行したり文書を開いたりしないこと、データ等を外部から取り込む時にウィルス防御ソフトウェアでチェックすることなどが挙げられる。
- スパイウェアによる情報流出 — 有益なソフトウェアのふりをしてユーザに利用させ、その過程でこっそりユーザの情報を外部に送信するなどの動作を行うものをスパイウェアと呼ぶ。ウィルスと同様、感染機能を持つものもある。防御策としてはウィルスと同様、素性の分からないプログラムを動かさないことに尽きる。

- 不正侵入 — ネットワーク経由で他人が所有するコンピュータの利用権限を取得すること (これは犯罪行為)。侵入者は権限を取得したあと、そのコンピュータを他のサイトをさらに攻撃したりする「踏み台」に使うことが多い。対策としては、コンピュータを攻撃しやすい状態に置かない。具体的には、アカウントにはきちんとパスワードをつけ、セキュリティホール (抜け穴) がないようにシステムの更新を絶えず行い、ネットに接続する場合は最小限の通信のみに制限したファイアウォールの内側に置く。
- 架空請求 — 手当たり次第に「あなたは○○を利用したので料金を払ってください、さもないと法的手段に訴えます」などの文面の連絡を送り付け、だまされた人が入金するのを待つ。詐欺罪に相当する。対策は無視するか、悪質なもの (または正規の裁判所がからむもの) については警察や消費者センターなどに相談する。
- フィッシング — カード会社や銀行などを装って「アカウントが更新になりますので情報を入力してください」などと書かれた Web ページ (本物らしく見えるように作っておく) に誘導し、クレジットカード情報などを不正に取得する。対策としては、Web ページからそのような情報を更新させることはないので無視するか、不安なら電話など他の手段で問い合わせる。
- 出会い系サイト — **出会い系サイト**とは交際相手を求める男女のためのサイトをいう。出会い系サイトを介して児童・生徒が性犯罪などのトラブルに巻き込まれる事件が数多く発生しており、社会問題となっている。出会い系サイトの存在そのものは犯罪とは言えないし、以前から児童生徒が性犯罪に巻き込まれるケースも存在はした。しかし、携帯電話やネットという身近に利用する手段を通じてアクセスでき、しかも誰もが容易に身分を偽って (たとえば生徒仲間であるかのように) ふるまえるため、児童・生徒がトラブルに巻き込まれる温床となっている。このようなサイトには近づかず、このようなサイトを通じたコンタクトがあれば保護者や教師に相談するなどの注意が必要である。

情報システムの障害

今日の我々の生活は多くの情報システムに深く依存している。このため、これらの情報システムに障害があって使えなくなった場合に、重大

な影響を被る可能性がある。具体的な可能性として、次のようなものが挙げられる (ほんの一例):

- 金融機関の取引システムが停止した場合、必要な決済ができず、巨額の損害を被ったり、最悪の場合は倒産などに至る場合がある。
- 物流企業のシステムが停止した場合、企業が生産した商品の送付ができず、販売機会を失って損失を出す可能性がある。また原材料等が搬入できないため企業の生産活動が停止してしまうことがある。
- 工場などの制御システムに障害があった場合、生産活動が停止する可能性がある。もっと運が悪い場合、機器や装置の適切な制御ができず、爆発事故や有害物質の漏洩事故に至ることもある。
- 航空機などの制御システムに障害があった場合、運行ができなくなる。もっと運が悪い場合、運行中の不意の障害により事故が起きたり人名が失われたりすることもある。

情報システムの障害が事故につながらないようにするため、障害が起きた場合でも処理能力を縮小した状態で動き続けられるように設計したり (**フェイルソフト**)、停止する場合でも事故が起きないような形で停止できるように努める (**フェイルセーフ**)。たとえば、信号の制御が停止する場合でも、すべての信号が「赤」で止まるようにしておけば、通り掛かった車両は停止するのでただちに事故につながる可能性が小さくできる。

しかし、システムの種類によるがフェイルセーフが難しいものもある。たとえば地上の交通機関であればとりあえず止めれば衝突が防げるが、航空機などでは「止める」わけに行かない。また化学プラントなどでもすべての動作を止めただけでは装置に残った原料の反応が続いて爆発に至る場合もあり、徐々に反応量を減らしたり冷却を続けたりなどの動作が必要である。

また、情報システムなどに含まれるソフトウェアはその性質上、トラブルが起きた時に動作する部分は正常状態では使わない部分なので、その部分に誤りが含まれていて、いざトラブルににになってその対処が必要なときにその誤りが発現してうまく動作しなくなることがある。

実際には、情報システムでトラブルが起きた時に最後は人間の指示による対処が必要になることも多い。しかし、もともとは人間がこなしていた業務であっても、情報システムが作られて自動化することで人間に

は扱い切れないような大規模な処理を行うようになっていたり、自動化にまかせることで人間がその業務を普段行わなくなった結果、事故の時に引き継ぐ能力が失われているなどの問題で、人間の対処がうまく行かなくてトラブルが長引くこともある。

課題 実際に情報システムに障害が発生し、社会的な問題になった場合を調べてみよう。そのような障害が発生する原因にはどのようなものがあるか考えてみよう。我々はどのような対策を取ることができるか考えてみよう。

個人への影響

社会全体の情報化が進むにつれて、事故や犯罪以外の日常場面でも、個人に対してさまざまな情報化の影響が及ぶようになってきている。どのようなものがあるか、考えてみよう:

- 省力化と失業問題 — 情報機器の発達により、人間が不得手な単調な繰り返し作業や危険な作業を、コンピュータやそれに制御された機械によって行うことができるようになった。これによって、それまで人間が行っていた様々な仕事を、コンピュータや機械が行うようになってきた。これに従ってそれまでこのような仕事を行っていた人々が失業してしまうことになった。産業革命でも失業問題は存在したが、コンピュータとネットワークの普及によって、産業革命では問題にならなかった事務作業員 (ホワイトカラー) の失業も発生するようになった。
- 情報弱者 — 多くの人がコンピュータやネットワークを使いそこを流通する情報を活用するようになると、高齢などの身体的事情や経済的事情などからネットワークの情報を参照できない人にとっては他の人が得られる情報が得られずに不利を被ることが起きてきた。このような問題を**情報弱者** (ディジタルディバイド) の問題と呼ぶ。
- 身体的/精神的影響 — 仕事や趣味などでずっとコンピュータに向かっていることで目やその他の身体部位だけが酷使されたり、逆に動かないため運動不足になったりして健康が損なわれることがある。また、他人との会話や気分転換が少なくなることで精神的な健康にも問題が出ることがある。ネットワーク上のやりとりに

のめり込みすぎて現実の人との関わりが苦手になる場合 (**ネット中毒**) もある。

社会の変化とそれに伴う問題はいつの時代にもつきものであるし、どれも簡単な解決方法があるようなものではないが、このような問題について常に意識し、必要ならそれを軽減するよう配慮することは必要だと言える。

課題 情報化社会の個人へのマイナスの影響に対して、我々はどのような対策を行えるだろうか? 考えてみよう。また、行政などによる支援策があるかどうかも調べてみよう。

4.3.3 情報社会における安全性

安全性 (**セキュリティ**) とは一般に、危険や危害に会わないことをいう。情報社会における安全性といった場合は、情報社会や情報技術にかかわるもろもろの問題 (たとえばサイバー犯罪など) から身を守ることを言う。

サイバー犯罪とそれに対する防御については既に述べたので、ここではそれ以外の安全性阻害要因とその防御策について挙げておこう:

- 情報の漏洩 — 不正侵入などがなくても、データにアクセスできる状態のまま席を立ってしまうなどにより情報が漏洩することがある。個人情報をそれと気づかないまま Web などで公開してしまうこともある。これらの問題については、どのような情報が保護されるべきか予めリストアップして注意するのがよい。パソコンなどを廃棄するときは、データを保管してあったハードディスクを専用ソフトウェアで完全消去するか、物理的に破壊するなどの注意が必要である (単にファイルを消しただけだとデータ本体は残っていて復活できることがある)。
- データの消失 — 機器の故障や自然災害などにより、重要なデータが失われてしまうことへの防御も必要である。それには、定期的にデータの写し (バックアップ) を取り、安全なところに保管することが挙げられる (写しを盗まれることも情報漏洩になるので保管方法も十分考慮する必要がある)。

- 機器の障害 — 作業に必要なときに機器が稼働しないなどの問題も重要な作業であれば対策が必要となる。停電に対処するには無停電電源装置を使うことができる (停電時に作業中のデータを安全に保存する余裕が持てるという点でも望ましい)。機器の故障に対しては、必要なら予備の機器を準備しておくことが考えられる。

4.4　情報社会と個人

4.4.1　情報社会を生きる上で

今日の情報社会においては、一人の個人が他の人と交流できる範囲は飛躍的に広まっている一方で、それぞれの人との関わりは希薄になっているといえる。これは、情報社会が持つ次のような特性によっている:

- Web ページやブログなどの手段によって、これまでは不可能だった「個人が世の中の多くの人に対して情報発信すること」が可能になり、簡単に多くの人にメッセージを伝えられるようになった。
- ネットワークの発達により、距離的にどんなに離れていても、容易にメッセージをやり取りし合える。また、掲示板などの手段を通じて、これまでにない広い範囲の人と情報を交換したり知合いになったりできる。
- 一方でネット上のやり取りは言葉だけのものであり、共通の生活基盤や日常の場を持たないため、相手が実際にどのような人なのか等、周辺のことがらは分かりにくくなっている。
- ネット上のコミュニケーションに時間を割くぶんだけ、現実の人間との交流は希薄になりがちであり、また現実の人間であっても表面的なやり取りしかしない場面が増えている。

これらのことを考えるなら、過去におけるよりも一層、個人がそれぞれ自分の主張やポリシーを明確に意識し、「あうんの呼吸」ではなく明確に自分の言葉に出して説明していくことが必要になると言える。

またこのとき、より多様な相手とうまくつき合うためには、自分からは常に言葉づかいを丁寧にし、相手のさまざまな言葉づかいに対しては寛容になるなどの姿勢も必要になる。

コミュニケーションをスムーズに進める上で「マナー」「エチケット」が大切ということはよく言われるが、これは言い替えれば多様な相手

に対して寛容になり、相手の気持ちを思いやって行動することに他ならない。

演習 自分は人生において何を目標として持つのか、またその目標に照らして、他人とやりとりするときは何を目的としているのかを書き下してみよ。

4.4.2 情報社会における約束ごと

情報社会における法規制

「廊下を走っちゃいけないんだよ」「そんな法律でもあるのか?」という言い争い (?) に見られるように、法律は我々が「してはいけないこと」を絶対的に決める基準であるかのように扱われている。

しかし法律も、元々は「人のものを取ってはいけない」「人に暴力をふるってはいけない」のような常識や約束ごとから始まってそれを厳密に条文化したものである (もちろん、社会として禁止したり白黒をつけなければならないような内容のことがらに絞っての話である)。

我々の日常生活に関わる法律は長い間を経て現在の形になったものなので、それなりに社会の「常識」を反映したものになっているが、今日の情報技術の急速な進化はこのような熟成を経ていないため、うまく法律で扱えていない部分がある。

たとえば、情報のなかには高い価値のものもあり、それを盗むことは犯罪であるように思えるが、日本の法律では窃盗罪は「形のあるもの」だけを対象としているため、窃盗罪を直接適用することはできない。パスワードなどで保護されているものを取得した場合は不正アクセス防止法違反に問えるが、たとえば企業の社員がもともと業務として扱っている情報を無断で持ち出すような場合はそれを直接罪に問うことができない。

もちろんこのような「不具合」は今後次第に手直しされて行くことになると思われるが、たとえそうなったとしても単に法規制がどうである、というだけでなく「何が護られるべきか」「それはなぜか」という原則に常に立ち帰って考えることも大切であり続けるだろう。

個人情報の保護

個人情報とは一般的な用語としては「個人に関する情報」を意味する。2005 年 4 月に通称「個人情報保護法」が施行され、一定の条件を満たす企業や組織では個人情報をきちんと管理し保護することが義務づけられるようになった。

しかし、個人情報そのものは昔からあったのに、なぜ今日これが問題になっているのだろうか? それは、情報技術の発達により、多数の個人情報を収集したり加工することが大変容易になり、悪用の危険も増大しているという理由による。個人情報保護法は、一定量の個人情報を扱う組織等に対して規制を掛けることにより、悪用の危険を軽減しようとするものだといえる。

個人の活動についてはこの法律の対象外であるが、その場合でも不用意に自分や他人の個人情報を明かしたりするべきでないのは当然である。その理由としては次のものが挙げられる:

- 個人に関わる情報には**プライバシーの権利**があり、本人以外が勝手に公開するべきでないという考え方が一般的である。
- 個人に関わる情報の中にはそれを悪用することで本人に危害が及ぶものがある (たとえば誕生日と保健証番号を使って当人になりすまして借金をするなど)。

たとえば、電話番号と住所と名前だけでも、ストーカが特定個人につきまとう上では十分な情報になってしまう。このため、Web ページなどで情報を発信するときも、個人情報に相当するものについては注意深く検討して必要なものだけを出すのがよいだろう。

知的財産権

先にも述べたように、情報は本来形がないにも関わらず、その中には一定の価値を持つようなものが存在する。このようなもののうち、人間の知的活動によって作り出されたものを**知的財産**と呼び、それに関わる権利を知的財産権と呼ぶ。

知的財産権のうち、音楽、絵画、文章など知的創造に関わるものの保護は著作権と呼ばれ、それ以外の主に経済活動などに関わるものは**産業財産権**と呼ばれる。産業財産権の代表的なものを挙げておく。

- **特許権** — 新しい種類の機械や仕掛けなど「発明」を保護する。
- **実用新案** — 「発明」よりは簡便な水準の「考案」(物の形や構造や組み合わせ方などのアイデア) を保護する。
- **意匠権** — 独自性のある物品の形や模様などを保護する。
- **商標権** — 商品やサービスに使用するマークを保護する。

たとえば、著名な企業や商品の名称は商標として登録されているので、これと同じ名前を同じ分野の物品につけて使用することはできない。上で「登録」という言葉が出て来たが、産業財産権の場合は申請を行って認められてのちに保護されるという形になる。

このため、たとえば自分が重要な発明をしたとしても、自分が特許申請を行わないままでいて他人が特許を取得してしまい、自分がそれを勝手に使えなくなるということも起こり得る。

著作権

著作権は音楽、絵画、文章など**著作物**に関わる権利であり、著作権法によって保護される。著作権法では著作物を「思想又は感情を創作的に表現したものであり、文芸、学術、美術又は音楽の範囲に属するもの」と定めている。また、コンピュータのプログラムも日本では著作物として扱われ、著作権法によって保護されている。

著作権法では芸術的価値がどうかということには言及しないので、自分が書いた絵や自分が書いた文章であっても、著名な画家や文筆家の創作物とまったく同等に保護される。また、著作権法による保護は申請を必要としないため、誰の作品でも創作した時点で自動的に保護される。これらの性質から、著作権は我々にとって関係の深い権利だと言える。

たとえば、あなたが Web ページを作ってそこに自分で書いた文章を置き、また自分で描いた絵を入れたとしよう。これらはあなたの著作物であるので、これを見た他人がそれを勝手に流用したとすれば、著作権法を侵すことになる。つまり、あなたが自分の文章や絵を勝手に使われているのを見てそれをやめて欲しいと思うなら、それを止めさせる権利があるわけである。

著作権は大きく次の 3 つに分かれている:

- 著作者人格権 — 著作者 (著作物を作った人) に付随する権利で、他人に譲渡できない。公表権 (いつ、どのように公開するか決める

権利)、氏名表示権 (どのように名前を表示するか決める権利)、同一性保持権 (勝手に改変されない権利) から成る。

- 著作財産権 — 著作物に関わる金銭的権利で、複製権、上演権、公衆送信権、譲渡権、翻訳権などが含まれる。
- 著作隣接権 — 著作物を伝達する者 (歌手や俳優などの実演家、レコード製作者、放送事業者などが持つ権利。

たとえば、著作権の中には複製権が含まれているので、著作者の許諾なしに他人が勝手に著作物を複製することはできない。テレビ番組 (これも著作物) をビデオ録画することも複製に相当することに注意。

ただし、ビデオ録画もできないのではあまりに不便である。もともと著作権は、著作者の権利を保護することを通じて文化の発展に寄与することを目的としているので、適正な著作物の利用もできないのではその主旨に反する。このため、個人が家庭内など限られた範囲で私的に利用するための複製は**私的利用**として認められている。多数の人にコピーを渡すなどの行為は (たとえお金を取らないとしても) 私的利用の範囲を超えているので著作者の許諾を得ない限り著作権法に違反することになる。

また、正規の学校においては、授業のために著作物を複製することもとくに認められている。このため、課外活動のために楽譜をコピーする、学校ではなく塾での授業のために著作物をコピーするなどの行為は著作者の許諾を得なければ著作権法に違反することになる。

もう 1 つ重要な概念として、**引用**がある。引用とは、著作物の一部を引き写してきて利用することであり、この時は著作者に断る必要がない。これは、文化の発展のためには著作物の批評や紹介などは必要な行為であり、その際に引用が必要と考えられるからである (たとえば引用するのに許諾が必要だとすると、批判的な批評がしにくくなるなどの問題が考えられる)。ただし引用するに当たっては、次の条件を満たす必要がある:

- 自分の作品が主であり、引用する部分が従であること。
- 必要最小限の範囲を引用すること。
- 出典が明示され、引用の範囲が明確に分かること。

引用の条件を満たさないで写してくることは**転載**に当たり、これは著作者の許諾が必要となる。

4.4.3 情報と職業

職業に対する考え方

我々の社会は、労働によって価値を生み出す人が一定数いて、その人たちの生み出す価値によって子供やお年寄りなど働けない人たちを含めたすべての人たちの生活を支えている。

わが国は資源に乏しい国であり、食料自給率もわずか (40%!!) と他の先進国に比べて飛び抜けて低い (たとえば米国は 125%つまり食料輸出国である)。このため、労働者が価値を生み出し、それを金銭に換えて食料を輸入して行かないと、食べて行くことができないわけである。このことからも、働くことの必要性は明らかである。

では、働くことは必要に迫られて嫌々することなのだろうか。何に価値を見出すかは人によって様々だが、自分の労働が他の人に役に立つ事、または自分の労働で生み出される成果物に価値を見出し、働くことを生きがいの一部としている人は多数いる。

もちろん職業以外の趣味や社会活動などに価値を見出す人も多いし、それはそれで好ましいことであるが、それに加えて自分に適した職業に就き、働くことにも一定のやりがいが見出せれば、その方がその人にとってより幸せなはずである。

「情報」に関わる職業は、多くの資源を必要としないため、資源の少ないわが国においても他国と比べてハンディキャップを負わずに済むという利点がある。また、その多くが知的生産物としての「情報」を生み出すという側面を持つため、「自分独自のものが作り出せた」という満足を得やすい。そして、情報社会の今日では「情報」に多くの価値が見出されるため、自分の生産物に対して比較的高い対価を得やすく、またよい社会的評価を得やすいという特徴がある。このような特徴を持つ「情報」に関わる職業としてどのようなものがあるか、以下で見てみよう。

クリエータ

クリエータは「価値を持つ情報を生み出す」職種であり、情報社会の今日では非常に多くの需要が存在している。

伝統的なものとしては、画家、文筆家、ライタ、作詞家、作曲家など独立して仕事をする職業がこれに含まれるが、この分野でもテレビ・ラジオのシナリオ作家などメディアの発達に伴い新しい職種が生まれている。

また、音楽や台本などの作品を伝達するのには、歌手、演奏家、俳優などの職業が必要となるが、これは自営に近い形態もあるし、事務所に所属してマネジメントされている場合もある。

もっと多いのは企業などに所属するクリエータであり、イラストレータ、コピーライタ、各方面のデザイナやコーディネータなど多くの職種がある。とくに広告産業、被服産業、リビング産業、建築産業などは顧客の「好み」に応えることが重要な業務なので、多くのクリエータが活躍している。また能力があれば独立して仕事をすることもできる。

メディア業界

新聞・出版・テレビ・ラジオ・映画などのメディア業界は「情報」を発信すること自体によって利益をあげるので、そこには多くの「情報」に関わる職種が存在している。

新聞・テレビ・ラジオであれば報道する情報を取材する記者が必ず必要であるし、新聞･雑誌･本などでは紙面に適切に情報を配置するために編集者が必要である。テレビやラジオでは情報を伝達する職業としてアナウンサがある。また、番組や映画を製作するにはプロデューサ、ディレクタ、音声、カメラマン、編集など多く業種が関わっている。

情報技術者

ひとくちに**情報技術者**といっても、その中には非常に多くの職種がある。たとえば、企業などの大規模な情報システムを開発する仕事の場合、プログラミングの作業がメインになると考えるかも知れないが、実際には「どのような業務をどのようにシステム化するか」決める部分が非常に重要となり、プログラムを書く部分の比重はさほど大きくない。そして大きなシステムの場合多人数での分業になるため、担当箇所によって作業内容は大きく違って来る。また、システム開発以外にも情報技術者にはさまざまな仕事がある。

このため、土台の知識としてコンピュータの原理やプログラミングについて理解していることは必要だが、その先の知識や仕事の内容は職種によってまちまちである。逆に言えば、「情報技術者」の中にはそれぞれの人の特性を活かせるさまざまな職種があると言える。

情報処理技術者にどのような技能 (スキル) が求められるかについては、ITSS と呼ばれるスキル標準が参考になる:

http://www.itss-japan.com/contents/ITSS.html

また情報処理技術者の水準を評価する「情報処理技術者試験」にどのような分野が含まれているかも参考になる:

http://www.jitec.jp/1_11seido/seido_gaiyo.html

オフィスワーカ

オフィスワーカはホワイトカラーとも呼ばれ、職場で机に向かって仕事をする人全般を指す言葉である。その職務は計画、企画、経理 (会計)、営業 (販売)、など極めて多様であるが、いずれも「情報」を扱っている点では共通している。

今日では多くの企業において、厳しい競争に勝ち抜くため、「これまでの仕事のやり方どおり」を見直して、より多くの価値を生み出せるようなやり方を模索している。このため、オフィスワークにおける情報の流れや扱い方にも新たな創造性が求められるようになっている。

また、企画や販売などの業務においては、自分の考えたプランや販売している商品などの特徴を、経営者や顧客に効率よく伝えて納得を得ることが重要になる。このような場面での論拠となる情報の収集や情報の伝達方法には多くの工夫の余地があるといえる。

第II部

情報II

第5章　コンピュータとプログラミング

5.1　コンピュータの動作とプログラム

5.1.1　なぜプログラミングについて学ぶのか?

コンピュータと情報社会

現代は**情報社会**だと言われるが、その要点は大きく分けて次の 2 つだと言えるだろう (図 5.1):

- 情報に多くの価値が置かれ、人々が情報の生産、流通、消費に多くの時間を費している。
- 大量の情報を個々の人の要求に合わせて遠隔地まで流通させたり自在に加工することが技術的に可能になり、また実際に行われている。

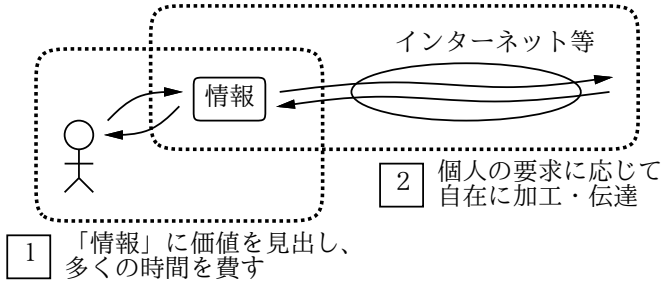


図 5.1: 情報社会の 2 つの特性

これらは互いに「にわとりと卵」の関係にある。つまり、前者のようなニーズがあってはじめて後者のような技術が普及するという面もある

し、後者のような技術がなければ前者のようなライフスタイルは成立し得ないという面もあるだろう。ここで、後者の技術の中核には**コンピュータ**があることに注意したい。

コンピュータと自動的な処理

過去においても情報を大量に複製したり (例:印刷技術)、遠隔地まで流通させたり (例:電話)、その両方を行う (例:ラジオ・テレビ) 技術は存在していたが、「何を」「どこに」「どのように」という部分については人間の介在が必要だったし、そのため「個人が自分の好きな時に好きなように」というわけには行かなかった (図 5.2 上)。

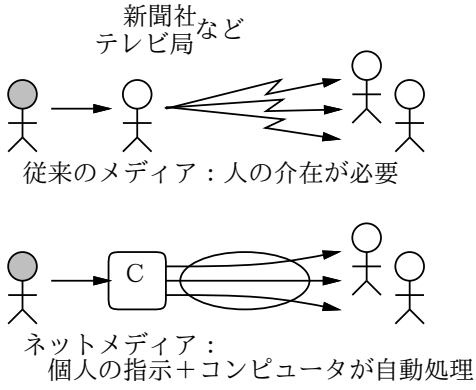


図 5.2: 従来のメディアと新しいメディア

しかし、コンピュータであれば必要な加工や転送を行う**手順**を**ソフトウェア**という形で用意してさえあれば、あとは人間の介在なし (**自動的**に) にこれらの作業をこなすことができる。つまり、各個人が「これをやりたい」と考えた時にそれをコンピュータに伝えさえすれば、誰かの手をわずらわせたり誰かがやってくれるのを待つことなく、組み込まれた手順によって直ちにそれが行えるわけである (図 5.2 下)。

ただし、そのような情報技術の恩恵にあずかるためには、その「使い方」と「原理・特性」の両方を知っておく必要がある。「使い方」については、自分がやりたいことをコンピュータに伝えるために必要だということは分かる。

では、「原理・特性」についてはどうだろうか。コンピュータをはじめとする現在の情報技術にはできることとできないことがあり、「原理・特性」を知っていてはじめて、自分がコンピュータに依頼したことがどのように行われ、どのような結果をもたらすかを的確に判断できる。そして、「自分一人の要望に基づいて」「ただちに」情報の伝送・加工が行われるということは、自分が間違ってもそれを止めてくれる人がいないことを意味するわけなので、各自が的確な判断能力を持つことは極めて重要なのである。

このため、以下ではコンピュータの動作原理とその特性について学んで行く。また、その過程を通じて、コンピュータに**プログラム**という形で指示を与えることについても学ぶ。これを知っておくことで、コンピュータに自分が作り出した手順を教えることができ、**自分が介在しなくても**自分の意図したことをコンピュータに行わせることができるようになり、より高度な形で今日の情報技術を使いこなせるようになるはずである。

5.1.2 コンピュータと手順

コンピュータと電卓の比較

ここで、もっとも手軽な「計算する道具」である**電卓**を題材にしながら、コンピュータの動作原理を学んで行くことにしよう (図 5.3)。

コンピュータでも電卓でも、計算をするにはまずそのデータを**入力** する装置 (**入力装置**) が必要である。電卓ではこれは数字キー、コンピュータではキーボードやマウスなどに対応する。また、計算結果を最後は人が使うので、人が結果を読み取れるように**出力**する装置 (**出力装置**) が必要である。電卓ではこれは表示窓、コンピュータではディスプレイ装置 (画面) などに対応する。

では、電卓とコンピュータのもっとも違うことは何だろうか? 電卓では確かに計算はできるが、その計算をする手順は人間が逐一、打ち込んでやらなければならない。これに対し、コンピュータでは計算手順をプログラムという形で予め蓄えておくことで、人間の介在無しに (自動的に) 処理を進めることができる。

この、「演算する」部分と「手順を制御する」部分はコンピュータの**中央処理装置** (**CPU**) と呼ばれる部品に含まれている。また、自動的に

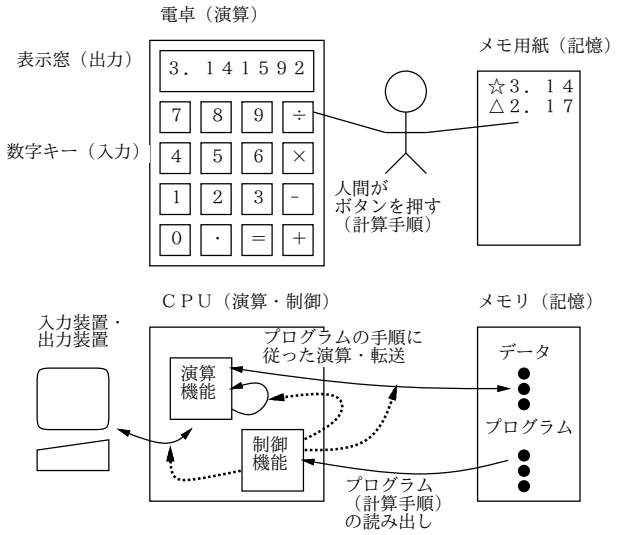


図 5.3: 電卓とコンピュータ

計算を進めるには、その計算に使うデータやプログラムを保管し、高速に出し入れできる**メモリ (主記憶)** が必要である。

電卓の場合は、計算手順は人間がボタンを押すことで実現し、またデータの保管は人間が必要のつどメモ用紙などに記録したり、メモリ機能を使って保存するなどの指示をおこなう。

計算と手順

もう少し、電卓による演算を詳しく見てみよう。たとえば、底辺が 15cm、高さが 8cm の三角形の面積を求めるとしたら、次のようにキーを打つことになるだろう:

1	5	×	8	÷	2	＝

では、底辺が 22cm、高さが 6cm の三角形ではどうだろうか:

2	2	×	6	÷	2	＝

底辺が 9cm、高さが 16cm だとどうだろうか:

9 × 1 6 ÷ 2 =

どの場合でも、「計算のしかた」は同じだから、計算するために押すキーは同じである。しかし、「計算するデータ」は三角形が違えば違ってくる。そこで、毎回代わるデータのところを☆、★などで表してみよう。

☆ × ★ ÷ 2 =

こうしておけば、たとえば☆、★のところで人間が計算したい三角形のデータを打ち込み、あとはたとえばロボットが決まったキーを押すようにして「自動的に実行」することで、何回でも間違いなく三角形の面積を計算できるし、人間の方は面積の計算のしかたを忘れてしまっても大丈夫である。

このように、手順をデータから分けて取り出し、仕組みを知らないでも何回でも間違いなく実行できるようにしたものがプログラムなわけである。

5.2 JavaScript によるプログラミング

5.2.1 入力・処理・出力

JavaScript 言語と HTML

以下ではプログラミング言語の 1 つである **JavaScript** 言語を用いて、実際にプログラムのさまざまな側面を学んで行く。JavaScript は、Web ページに組み込んで実行するために作られた言語であり、一般的に使われているブラウザには JavaScript の言語処理系が組み込まれている。

ブラウザに JavaScript プログラムを読み込ませて実行させるには、プログラムを **HTML**(HyperText Markup Language、Web ページの内容を記述する言語) の中に埋め込む形で書く必要がある。ここでは、最低限の HTML として次の形のものを書くことにする。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html><head><title>sample</title></head><body>
<script type="text/javascript">
... JavaScript のプログラム ...
...
</script>
</body></html>
```

このような内容のファイルを「なんとか.html」という名前をつけて作成し、それをブラウザで開くことで JavaScript プログラムを実行させられる。

例題: 2 つの数の和

最初の問題は次のものとする:

- 2 つの数値を読み込み、その和を計算して、出力する。

多くのプログラムは、処理するための情報を外部から受け取り (入力)、その情報を加工し (処理)、結果を外部に返す (出力)、という段階を踏んで動作する。この例題はそのようなほぼ最小限の構造を持ったものだといえる。

JavaScript でできるだけ簡単に入力を取得する方法としては、「`prompt(…)`」により画面にメッセージを表示してユーザに文字列を入力してもらうものがある。その結果は文字列なので、数値が必要なら「`parseFloat(…)`」を用いて数値に変換する。

JavaScript でできるだけ簡単に画面に出力する方法としては、「`document.write(…)`」により文字列を書き出すものがある。書き出したものはブラウザが表示する HTML の`<script>...</script>` の位置に挿入された形で現れる。[1]

プログラム言語での記述には細かい制約があるため、最初から**コード** (プログラム言語による記述) を書きはじめてしまうと処理の見通しが分からなくなってしまうことがある。このため、プログラムで何をどのように処理するかの手順を日本語などで記述するのがよい。これを「コードに似ているが、完全にコードというわけではない書き方」という意味で**擬似コード**と呼ぶ。上の例題の擬似コードを示す:

```
数 1 を入力し、その結果を変数 x に入れる。
数 2 を入力し、その結果を変数 y に入れる。
x と y の内容を足し、結果を変数 z に入れる。
z を画面に書き出す。
```

[1] ブラウザによるページの表示が終わった後で `document.write()` を使うと、いったんブラウザの表示内容はすべて (JavaScript プログラムまで含めて) クリアされ、それから内容が書き始められるので注意が必要である。

ここで**変数**とは、プログラミング言語では「値を入れておくことができる、名前つきの場所」を表し、数学でいう「変数」とは意味が違っている (前節でメモリ上の番地に a、b などの名前をつけて表しているが、それと同様に考えるのがよい)。

JavaScript では、変数を使うときは「var 変数名;」によって変数を使うことを**宣言**する。「var 変数名 = 初期値;」で最初の値を入れることもできる。[2]

ここまででだいたい準備ができたので、実現に進むことができる。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html><head><title>sample</title></head><body>
<script type="text/javascript">
var x = parseFloat(prompt(’数 1 を入力’));
var y = parseFloat(prompt(’数 2 を入力’));
var z = x + y;
document.write(’合計は’ + z + ’です。<br>’);
</script>
</body></html>
```

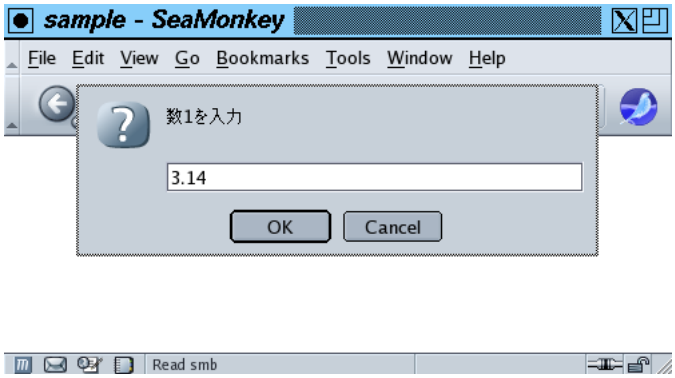


図 5.4: 2 つの数の和:入力しているところ

「’数値 1 を入力してください’」のように、文字を「’’」で囲んだものは**文字列** (文字が並んだデータ) を表す。「prompt(...)」は、メッセージ (「...」の部分) を画面に表示して文字列を入力してもらい、そ

[2]変数の宣言をしなくても動くことがあるが、どこで変数を使い始めるかきちんと書いてある方が、後で分からなくなりにくいので好ましい。

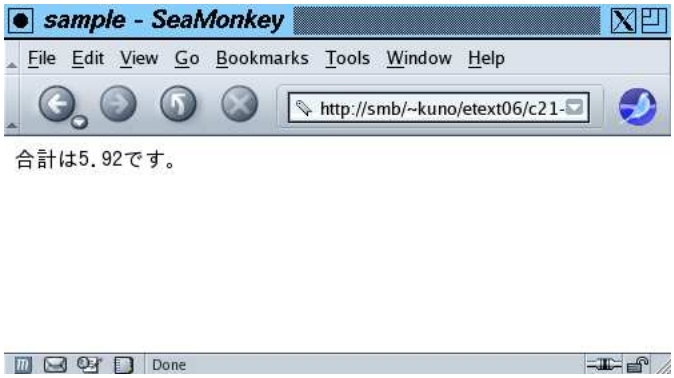


図 5.5: 2 つの数の和:結果を表示しているところ

の文字列を返す機能を表す。`parseFloat(...)` は渡された文字列を数値に変換する (たとえば「`3`」「`.`」「`1`」「`4`」の 4 文字から「3.14」という数値を作って返す)。

「変数 `=` 計算式`;`」は、右辺の計算式が計算した結果を左辺の変数に格納することを意味する。これを**代入**と呼ぶ。プログラムの 2 行目と 3 行目ではそれぞれ、入力された文字列を数値に直したものを変数 `x` と `y` に代入している。4 行目では変数 `x` と `y` に格納されている数値の和を計算し、その結果を変数 `z` に代入している。これらはいずれも「`var`」で始まり、変数の宣言を兼ねている。

JavaScript では、計算式の中では次の演算子を使うことができる。

- x `+` y — x と y の和を返す。ただし、x、y のいずれか一方が文字列なら、他方も文字列に変換した上で 2 つの文字列の連結を行い、結果の文字列を返す。例: `1 + 2` → `3`、`'abc' + 1` → `'abc1'`。
- x `-` y、`-`x — x と y の差、および x の符号を反転した数値を返す。例: `1 - 3` → `-2`、`-(-3)` → `3`。
- x `*` y — x と y の積を返す。例: `2 * 3` → `6`。
- x `/` y — x を y で割った商を返す。例: `7 / 2` → `3.5`。[3]
- x `%` y — x を y で割った時の余り (**剰余**)。例: `7 % 3` → `1`。

[3] 多くのプログラミング言語では、整数どうしの割算は結果も整数になるが、JavaScript では結果は小数点つきの値になることもある。

「`document.write(…)`」は、文字列 (「…」の部分) をページ内容としてブラウザ上に表示させる機能を表す。「`+`」は左辺または右辺が文字列の場合は、文字列の連結を表す。たとえば z に「`2.78`」が入っていたとすると、4 行目ではまず「`'` 合計は `2.78` です。`<br>'`」という文字列が作られ、それがブラウザで表示される。なお、「`<br>`」はブラウザに行かえを指示する HTML のタグである。

演習: 次のような JavaScript プログラムを書いて動かせ。

- 2 つの数を入力し、その和、差、積、商を出力する。
- 円の半径を入力し、その面積を出力する。
- 円錐の底面の半径と高さを入力し、その体積を出力する。

5.2.2 枝分かれ

例題: 2 つの数の大きい方を出力

2 番目の例題は、次のものとする:

- 2 つの数値を読み込み、大きい方を出力する。

この記述は一見よさそうだが、十分に厳密だろうか。たとえば、これでは 2 つの数値が等しかったときについては、何も書かれていないため、動作が決められない。そこで次のように記述を書き直すことにする:

- 2 つの数値を読み込み、大きい方を出力する。2 つの数値が等しい場合は、その数値を出力する。

先に学んだプログラムは、上から順番に動作を実行し、一番最後の動作を実行すると終わりというものだった。しかし場合によっては、プログラムの中で条件判断を行い、その成否に応じてプログラムの一部だけを実行したいこともある。このような構造を**枝分かれ**と呼ぶ。この問題は、枝分かれを必要とする可能性がある。

枝分かれの記述

枝分かれを表す擬似コードとしては、次の形のものを使う:

```
もし ～ ならば、
  動作 1…
そうでなければ、
  動作 2…
枝分かれ終わり。
```

ここで「～」は条件を表し、その条件が成り立っていれば「動作 1」、成り立っていなければ「動作 2」が実行される。条件が成り立っていない場合には何も動作が必要ないこともあるが、そのときは「そうでなければ、」と「動作 2」は書かなくてよい。

JavaScript では、枝分かれは **if 文**を使ってあらわす。その形は次のようになる (右側は「動作 2」がない場合):

```
if(条件) {
  動作 1…
} else {
  動作 2…
}
```

```
if(条件) {
  動作 1…
}
```

条件としては「x は y より大きい」などの比較演算を記述することが多い。JavaScript での比較演算は次の 6 種類がある。[4]

- `x == y` — x と y が等しい (「=」が 2 つ必要。1 つだと代入を表す)。
- `x != y` — x と y が等しくない。
- `x > y` — x が y より大。
- `x >= y` — x が y 以上。
- `x < y` — x が y より小
- `x <= y` — x が y 以下。

「2 つの数の大きい方を出力」の問題に戻って、その手順を見てみよう。たとえば次のようなものはどうだろうか:

```
数 1 を入力し、その結果を変数 x に入れる。
数 2 を入力し、その結果を変数 y に入れる。
もし ~ x ≧ y ~ なら、
  x を画面に書き出す。
そうでなければ、
  y を画面に書き出す。
枝分かれ終わり。
```

[4] 厳密にはほかに===と!==があるが、特別な場合にしか使わないので説明は略した。

これを JavaScript に直したものは次のとおり:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html><head><title>sample</title></head><body>
<script type="text/javascript">
var x = parseFloat(prompt(' 数 1 を入力'));
var y = parseFloat(prompt(' 数 2 を入力'));
if(x >= y) {
  document.write(' 大きい方の値は' + x + ' です。<br>');
} else {
  document.write(' 大きい方の値は' + y + ' です。<br>');
}
</script>
</body></html>
```

しかし、同じことは次のような手順でもできる:

```
数 1 を入力し、その結果を変数 x に入れる。
数 2 を入力し、その結果を変数 y に入れる。
もし ~ x ≧ y ~ なら、
  z ← x
そうでなければ、
  z ← y
枝分かれ終わり。
z を画面に書き出す。
```

この方法は、大きい方の値が変数 z に格納されるので、出力した後さらにその値を使いたい場合などにはこちらの方がよさそうである。これを JavaScript に直したものを示す:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html><head><title>sample</title></head><body>
<script type="text/javascript">
var x = parseFloat(prompt(' 数 1 を入力'));
var y = parseFloat(prompt(' 数 2 を入力'));
var z;
if(x >= y) {
  z = x;
} else {
  z = y;
}
document.write(' 大きい方の値は' + z + ' です。<br>');
</script>
</body></html>
```

変数 z の宣言 (var z;) は、動作ではないので擬似コードには現れていないが、プログラム上はこの先で z を使うので if 文の前に書いておくのがよい。

ところで、3 番目の方法として、「とりあえず z に x を入れておき、もしも y の方が大きいなら z に y を入れ直す」という形で枝分かれを使うことも考えられる:

```
数 1 を入力し、その結果を変数 x に入れる。
数 2 を入力し、その結果を変数 y に入れる。
z ← x
もし ~ y ＞ z ~ なら、
  z ← y
枝分かれ終わり。
z を画面に書き出す。
```

この方法でも、最後に z に大きい方の値が入っているという点は 2 番目の方法と同じである。これを JavaScript に直したものを示す:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html><head><title>sample</title></head><body>
<script type="text/javascript">
var x = parseFloat(prompt(' 数 1 を入力'));
var y = parseFloat(prompt(' 数 2 を入力'));
var z = x;
if(y > z) {
  z = y;
}
document.write(' 大きい方の値は' + z + ' です。<br>');
</script>
</body></html>
```

この方針による実現は、コード行数がやや少ないことと、比較する数が増えても複雑にならないという利点がある。

しかし、このような処理をいちいち自分で書かなくても、既にあるものを**再利用**できればその方が望ましいかも知れない。JavaScript には次のような計算をする**関数** (一連の計算をまとめて名前をつけたもの) が予め用意されている。

- Math.abs(x) — x の絶対値を返す。
- Math.floor(x) — x を整数に切り捨てた値を返す。

- `Math.round(`x`)` — x を整数に四捨五入した値を返す。
- `Math.ceil(`x`)` — x を整数に切り上げた値を返す。
- `Math.max(`x`, `y`, ...)` — 渡された引数群の最大値を返す。
- `Math.max(`x`, `y`, ...)` — 渡された引数群の最小値を返す。
- `Math.random()` — 0 以上 1 未満の一様乱数を返す。
- `Math.pow(`x`, `y`)` — x^y を返す。
- `Math.sqrt(`x`)` — $\sqrt{x}$ を返す。

これらのうち、`Math.max()` は今回の問題に利用できそうである:

```
数 1 を入力し、その結果を変数 x に入れる。
数 2 を入力し、その結果を変数 y に入れる。
Math.max(x, y) を画面に書き出す。
```

これを JavaScript に直すと次のようになる:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html><head><title>sample</title></head><body>
<script type="text/javascript">
var x = parseFloat(prompt('数 1 を入力'));
var y = parseFloat(prompt('数 2 を入力'));
document.write('大きい方の値は' +
               Math.max(x, y) + 'です。<br>');
</script>
</body></html>
```

このように、同じ動作をするプログラムでもさまざまな方針が可能なことが分かる。どのプログラムも問題の解という点ではすべて正しいが、それぞれに得失があり、条件によってどれを選択するか変わって来る可能性がある (どれでも構わないこともちろんある)。

このような複数の選択肢は、実際にはプログラムを作るという 1 つの問題を解決していくすべての段階で現れるものであり、それぞれの箇所で代替案を検討して適切に選択していく必要がある (どれを選択しても構わないこともあるが、その場合でもどれでも構わないことを確認してから 1 つ選ぶ)。

5.3　繰り返しと関数

5.3.1　繰り返し

例題: 三角形を描く

3 番目の例題は、次のものとする:

- 1 より大きい整数を読み込み、*を 1 行目には 1 個、2 行目には 2 個、…のように出力し、それを指定された数まで繰り返して「三角形」を描く。たとえば「5」を入力したら次のようになる。

```
*
**
***
****
*****
```

この問題では、「*が 1、2、3、…、N 個の行を出力する」ところで**繰り返し** (**ループ**) が必要になる。プログラムの中で繰り返しを使うと、その範囲のコードは多数回実行されることになる。コンピュータは繰り返しを使うことによって、「少ないコードでも大量の仕事をこなす」ことができるわけである。

繰り返しの記述

繰り返しの最も基本的な形として「指定した条件が成り立つ間の繰り返し」がある。これを次のような擬似コードで表す:

```
～ が成り立つ間繰り返し、
  動作…
繰り返し終わり。
```

この形の繰り返しは、JavaScript では **while 文**であらわす (条件の指定方法は if 文と同じである):

```
while(条件) {
  動作…
}
```

例題に戻って、繰り返しの中がどういう風になるかを考えてみよう。最初に変数 s に文字列「''」(長さ 0 の文字列) を入れておき、その後ろに「'*'」をつけ加えて 1 文字ずつ増やして行く方針にする。

```
s ← s + '*'。   // '*' になる
s を出力。
s ← s + '*'。   // '**' になる
s を出力。
s ← s + '*'。   // '****' になる
s を出力。
…
```

「+」は足し算ではなく文字列の連結を表している。上の記述を見ると、2 行ずつ同じ動作を実行しているので、それを繰り返しの中の動作とすればよいと分かる。繰り返しは、s が指定した長さ以下である間実行すればよい:

```
s ← ''。
s の長さが x 以下である間繰り返し、
  s ← s + '*'。
  s を出力。
繰り返し終わり。
```

文字列の操作

上の擬似コードを JavaScript にするためには、文字列の長さを調べるなどの機能が必要である。JavaScript で使える文字列の操作の代表的なものとして、次のものがある:

- $s1$ `+` $s2$ — 文字列 $s1$ と $s2$ を連結した文字列を返す (例: `'abc' + 'd'` → `'abcd'`)。
- s`.length` — 文字列 s の長さ (含まれている文字の数) を返す (例: `'abc'.length` → `3`)。
- s`.charAt(`i`)` — 文字列 s の i 文字目を返す。先頭の文字列を「0 文字目」として数えることに注意 (例: `'abcd'.charAt(2)` → `'c'`)。

- s.substring(i) — 文字列 s の i 文字目以降を取り出したものを返す。先頭の文字列を「0 文字目」として数えることに注意 (例: 'abcd'.substring(1) → 'bcd')。
- s.substring(i, j) — 文字列 s の i 文字目以降、j 文字目の直前までを取り出したものを返す。先頭の文字列を「0 文字目」として数えることに注意 (例: 'abcd'.substring(1, 3) → 'bc')。

これらの機能を活用して、上の擬似コードを JavaScript に直したものを次に示す (実行のようすは図 5.6、図 5.7 に示す):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html><head><title>sample</title></head><body>
<pre><script type="text/javascript">
var x = parseFloat(prompt(' 整数を入力'));
var s = '';
while(s.length <= x) {
  s = s + '*';
  document.writeln(s);
}
</script></pre>
</body></html>
```

なお、このプログラムではこれまでのプログラムと JavaScript の書き方で次の点が違っている:

- <script>...</script>が<pre>...</pre>の中に入れてある。
- document.write() の代わりに document.writeln() を使っている。

これは、「文字を使ったお絵描き」で空白文字や改行をそのまま表示させるために必要な設定である。以下の例はすべてこのような書き方を使っているものとする。

5.3.2　関数定義

ひとまとまりの処理と関数

プログラムの中で、ひとまとまりの処理に名前をつけたものを**手続き**と呼ぶ。手続きには、複数の箇所でその処理を必要とする時にそのつ

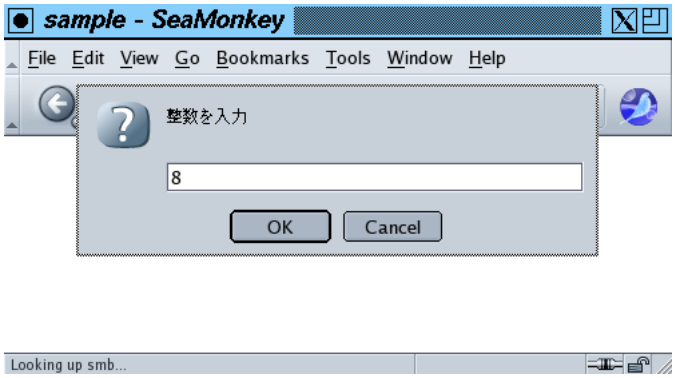


図 5.6: 三角形を描く:入力しているところ

どその処理を書かなくても、名前を指定して呼び出すことで 1 箇所に書いた処理を利用できるという利点がある。また (ある意味ではさらに重要なことだが)、ひとまとまりの処理に名前をつけてあるので、プログラムを作ったり読む時に把握しやすいという利点もある。

手続きのうちで、値を計算して返すようなものを (数学でいう関数になぞらえて) **関数** (function) と呼ぶ。JavaScript ではさらに拡大解釈して、値を返しても返さなくてもすべての手続きを関数と呼ぶ[5]。

JavaScript では、関数は次のような形で定義する:

```
function 関数名 (パラメタ名, ...) {
  関数本体...
}
```

ここで**パラメタ**とは変数の一種だが、関数を呼び出す箇所ごとに値を渡すのに使うことができる。また、「関数本体」の部分には一般の JavaScirpt の処理が記述できるが、加えて **return 文**が記述できる。

- `return 式;` — 式をこの関数の値として返し、関数の実行を終わる。
- `return;` — 何も返さずに関数の実行を終わる (とくに関数を値を返さない手続きとして利用する時に使う)。

[5]C 言語、C++言語、Java 言語の流儀にならったものである。

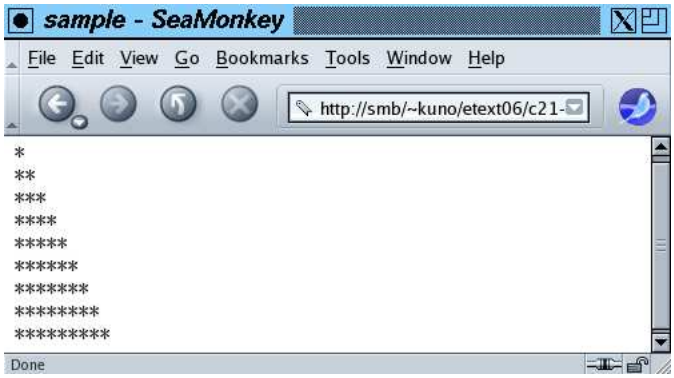


図 5.7: 三角形を描く:結果を表示しているところ

関数本体の文は関数の先頭から順番に実行されるが、最後まで実行してそれが return 文でない場合は「`return;`」が実行される。

たとえば、一次関数 $y = 2x + 3$ を計算して返す関数は次のようになる: [6]

```
funciton myfunc(x) {
  var y = 2 * x + 3;
  return y;
}
```

またはもっと短く、次のようにしてもよい:

```
funciton myfunc(x) { return 2 * x + 3; }
```

例題: 逆三角形を描く

4 番目の例題として、次のものを考えよう:

- 1 より大きい整数を読み込み、次のような逆三角形を描く:

```
*****
 ****
```

[6] ここで変数 y を var で宣言することは重要である。var なしに変数を使うと、それは関数の外側の変数と同じになるので、よそで使っている変数を間違えて書き換えてしまう可能性が出てくる。

```
***
 **
  *
```

ただし最初の*の個数は読み込んだ整数に等しいものとする。

これは 1 つ前の問題とそっくりに思えるかも知れないが、右側を揃える必要があるため、適宜空白文字を左側にくっつける必要があるので少し面倒になる。たとえば、上の例だと最初の行は「空白が 0 個、*が 5 個」、次の行は「空白が 1 個、*が 4 個」、その次の行は「空白が 2 個、*が 3 個」のように出力していく必要があるわけだ。

ここでかりに、次のような関数があったと考えてみる:

- `hanpuku(`c`,` n`)` — 文字 c(文字列でもよい) を n 回繰り返しつなげた文字列を返す。

このような関数があれば、上の問題の解となるプログラムの繰り返し (やはり繰り返しは必要だろう) の中身は次のようになるだろう:

```
hanpuku(' ', 0) + hanpuku('*', 5) を出力。
hanpuku(' ', 1) + hanpuku('*', 4) を出力。
…
hanpuku(' ', 4) + hanpuku('*', 1) を出力。
```

これから、全体は次のような擬似コードで表せる:

```
整数 x を読み込む。
i ← 0。
i < x である間繰り返し、
  hanpuku(' ', i) + hanpuku('*', (x-i)) を出力。
  i ← i + 1。
繰り返し終わり。
```

計数ループ

ところで、このループでは i を 0、1、2、…のように 1 個ずつ増やして数を数えるのに使っている。このような繰り返しを**計数ループ**という。計数ループはよく使うので、次のような専用の書き方をする:

```
変数○を□から△の手前まで変えながら繰り返し、
  動作…
繰り返し終わり。
```

そして、JavaScript ではこれを次のような形の **for 文**で書き表す (「++」は「変数を 1 増やす」という機能を表している):

```
for(var ○ = □; ○ < △; ++○) {
  動作…
}
```

これを使えば、上の文字列の擬似コードとその JavaScrpit 版は次のように短く書ける:

```
整数 x を読み込む。
変数 i を 0 から x の手前まで変えながら繰り返し、
  hanpuku(' ', i) + hanpuku('*', (x-i)) を出力。
繰り返し終わり。
```

```
x = parseFloat(prompt(' 整数を入力'));
for(var i = 0; i < x; ++i) {
  document.writeln(hanpuku(' ', i) + hanpuku('*',(x-i)));
}
```

これは大変簡潔で分かりやすいが、ところで実際には関数 hanpuku というのは存在していない。しかし、その機能は今までやった方法で実現できる。擬似コードと JavaScript コードを示そう:

```
hanpuku(c, n) : 文字列 c を n 回繰り返し連結して返す
  s ← '';
  変数 i を 0 から n の手前まで変えながら繰り返し、
    s ← s + c;
  繰り返し終わり。
  s を返す。
```

```
function hanpuku(c, n) {
  var s = '';
  for(var i = 0; i < n; ++i) { s = s + c; }
  return s;
}
```

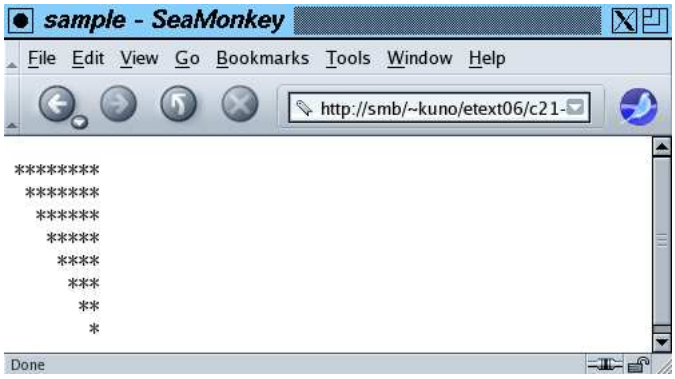


図 5.8: 逆三角形を描く:結果を表示しているところ

では、これと先のプログラムとを合わせた全体を示そう (実行結果を図 5.8 に示す):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html><head><title>sample</title></head><body>
<script type="text/javascript">
x = parseFloat(prompt(' 整数を入力'));
for(var i = 0; i < x; ++i) {
  document.writeln(hanpuku(' ', i) + hanpuku('*',(x-i)));
}
function hanpuku(c, n) {
  var s = '';
  for(var i = 0; i < n; ++i) { s = s + c; }
  return s;
}
</script>
</body></html>
```

このように、プログラムが複雑になってきたときも、ひとまとまりの機能を関数としてとらえ、その関数を作る作業と、関数を利用して残り

の機能を作る部分とに分けることで、全体の見通しを持って作業ができるようになる。

演習 次のような絵を描くプログラムを作成せよ。まず擬似コードを書き、次に JavaScript で記述すること。

a. ダイヤモンド形

```
   *
  ***
 *****
*******
  ***
   *
```

b. 次のような模様

```
+-+-+-
+-+-+
+-+-
+-+
+-
+
```

(ヒント) 最初に hanpuku() で長い「+-+-…」を作り、ループの中ではその先頭から何文字かを「s.substring()」で取り出すとよい。

c. そのほか、自分で描いてみたいと思った図形や模様。

第6章　アルゴリズムとデータ構造

6.1　アルゴリズムとその考え方

6.1.1　アルゴリズムと段階的詳細化

アルゴリズムとは

これまでに学んで来たように、コンピュータは、プログラムによって指定された手順をその通りに実行する。ある問題を解く手順が、次のような性質を持つとき、それを**アルゴリズム**と呼ぶ。

- 有限個の処理で記述できている。
- 各処理にあいまいさがない。
- 必ず結果を出して終了する。

コンピュータのプログラムは、それを書き終わった時点でその大きさは有限であり、各処理にあいまいさはない。これに加えて、必ず終了するように書かれているプログラムは、アルゴリズムを実現しているものと言える。

ただし、プログラムによっては必ず終了するとは限らない。たとえば、OS のプログラムはいつまでも実行し続けるように作られているし、プログラムに間違いがあって同じ計算を限りなく繰り返す状態になってしまうこともある。そのようなプログラムはアルゴリズムには対応していない。

アルゴリズムと一般化

アルゴリズムを記述するときは、直接プログラム言語でプログラムとして書くと、細かい処理が書かれているため分かりにくくなりやすい。

そのため、アルゴリズムを記述するときは、もっと抽象化された (大観的な書き方のできる)、人間に読みやすい記法を使うことが多い。ここでは、アルゴリズムを記述する方法として、前章に引続き**擬似コード**を用いる。

たとえば、「ある正の数 n を与えて、それを何回半分にしたら 1 より小さくなるか」という問題を考えてみよう。まず一般的なアルゴリズムを考えるまえに、具体的な数、たとえば「18」でやってみる:

```
18 / 2 → 9          // 1 回
9 / 2 → 4.5         // 2 回
4.5 / 2 → 2.25      // 3 回
2.25 / 2 → 1.125    // 4 回
1.125 / 2 → 0.5625 // 5 回
```

したがって、答えは「5」ということになる。

確かに答えは求まったが、これでは人間がすべての計算と判断をおこなっている。コンピュータに計算させ、最後に結果を打ち出すとしたらどうだろうか。それには、元の数「18」を変数に入れ、それを半分ずつにしていく、というふうにすればいいだろう:

```
x ← 18。
{x を半分にする。}  // 9
{x を半分にする。}  // 4.5
{x を半分にする。}  // 2.25
{x を半分にする。}  // 1.125
{x を半分にする。}  // 0.5625
{結果を出力する。} // 5 回
```

ここで、中かっこに囲まれた部分は、まだ概要を書いたレベルであることを表している。

しかし「結果」はどこで計算されるのだろうか? 「半分にした回数」が結果なのだから、別の変数を用意しておいて、半分にするたびに回数を 1 つず増やすことで「数える」必要があるわけである。

```
x ← 18。
count ← 0。
{x を半分にする。}  // 9
{count を 1 ふやす。} // 1 回
{x を半分にする。}  // 4.5
{count を 1 ふやす。} // 2 回
{x を半分にする。}  // 2.25
{count を 1 ふやす。} // 3 回
{x を半分にする。}  // 1.125
{count を 1 ふやす。} // 4 回
{x を半分にする。}  // 0.5625
{count を 1 ふやす。} // 5 回
count を出力する。
```

「結果を出力する」のはつまり変数 count の値を出力すればいい、というふうに具体的になったので中かっこを取った。

先に進もう。「18」は例として使った数であり、本来は x に値を読み込んで計算をするわけである。従って、次のようになるはずである:

```
数 x を読み込む。
count ← 0。
{x を半分にする。}  // ?
{count を 1 ふやす。} // 1 回
{x を半分にする。}  // ?
{count を 1 ふやす。} // 2 回
{x を半分にする。}  // ?
{count を 1 ふやす。} // 3 回
{x を半分にする。}  // ?
{count を 1 ふやす。} // 4 回
…
count を出力する。
```

このように、最初は具体的な例で考えておいて、後からそれを**一般化**することで、さまざまなデータに適用できるようなアルゴリズムを組み立てることができる。

ところで、確かにこのようにして計算はできるだろうが、「いつ」終わりにして結果を出力したらいいのだろうか。それには、まだ「半分にする」を続けるのかどうか判断する必要がある。となると、次のようにすればいいことになるだろうか:

```
数 x を読み込む。
count ← 0。
もし{x が 1 より大}なら、
  {x を半分にする。}  // ?
  {count を 1 ふやす。} // 1 回
  もし{x が 1 より大}なら、
    {x を半分にする。}  // ?
    {count を 1 ふやす。} // 2 回
    もし{x が 1 より大}なら、
      {x を半分にする。}  // ?
      {count を 1 ふやす。} // 3 回
      もし{x が 1 より大}なら、
        {x を半分にする。}  // ?
        {count を 1 ふやす。} // 4 回
        …
      枝分かれおわり。
    枝分かれおわり。
  枝分かれおわり。
枝分かれおわり。
count を出力する。
```

これでは「…」をどこまで続ければいいかは読み込む値によって変わってくるのでちょっと困る。

しかしよく見ると、上でやっていることは「x を半分にする」「count を 1 ふやす」というまったく同じ動作を何回も繰り返すことに他ならない。したがって、「条件が成り立つ間の繰り返し」に書き換えることができる:

```
数 x を読み込む。
count ← 0。
{x が 1 より大}である間繰り返し、
  {x を半分にする。}  // ?
  {count を 1 ふやす。} // ?回
繰り返しおわり。
count を出力する。
```

このように、繰り返しを使うことで、少量の記述でも大量の計算を行わせるようなアルゴリズムを組み立てることができる。

段階的詳細化

これで大筋はよさそうである。あとは、中かっこの部分を具体的な内容に置き換えていけば完成である:

```
数 x を読み込む。
count ← 0。
x > 1 である間繰り返し、
  x ← x / 2。   // x を半分にする
  count ← count + 1。 //count を 1 ふやす
繰り返しおわり。
count を出力する。
```

このように、アルゴリズムを考えるときは、最初は大まかに考えて記述し、次第に細かい部分まで正確にしてゆくのが考えやすい。これを**段階的詳細化**と呼ぶ。

この疑似コードを JavaScript にしたものも示しておく (<script>...</script> の内側だけ示すことにする)。

```
var x = parseFloat(prompt(' 数 x を入力してください。'));
var count = 0;
while(x > 1) {
  x = x / 2;
  count = count + 1;
}
document.write(count + ' 回半分にすると 1 未満です。<br>');
```

段階的詳細化でプログラムを組み立てて行くと、枝分かれや繰り返しが何重にも重なって理解が難しくなることがある。このようなときは、詳細化の内容を直接その場所に書く代わりに、関数を定義してその中に書き、もとの場所からは関数を呼び出すだけにする方が分かりやすい。

演習: 次のことを行うアルゴリズムを、疑似コードで記述してみよ。続いてその疑似コードが正しいことを説明してみよ。最後に JavaScript にして動かして確認せよ。

- 1 より大きい数 x を読み込み、1 に x を何回掛けると 1 億より大きくなるかを求めよ。

6.1.2　繰り返しを含むアルゴリズム

例題: 2 の平方根を求める

繰り返しを含んだアルゴリズムでは、「永遠に同じ計算を繰り返す」状態にならないように注意する必要がある。それには具体的にどのように考えればよいか学ぼう。

例として、ある x の関数 $f(x)$ の**根** (式 $f(x) = 0$ が成り立つような x の値) を 1 つ、求めるアルゴリズムを取り上げる。ただし、関数 $f(x)$ は次の条件を満足しているものとする。

- $f(a) < 0$ であるような a、$f(b) > 0$ であるような b が分かっている。
- $a < x < b$ の範囲で連続していて、かつ単調に増えて行く。つまり、x が大きくなるにつれて、必ず $f(x)$ も大きくなり、なおかつ値が「飛ぶ」ところがない。

たとえば、関数の例として $f(x) = x^2 - 2$ を考える。$a = 0$、$b = 2$ としてみると、$f(0) = -2 < 0$、$f(2) = 2 > 0$ で、$y = f(x)$ のグラフは右に行くほど y が大きくなるから、条件を満たしている。この範囲で $f(x) = 0$ の根を求めると、$x^2 = 2$ を満たすわけだから、$\sqrt{2}$ つまり 2 の平方根が求まるはずである。

では、アルゴリズムの疑似コードを示す:

```
a ← 0。
b ← 2。
|a - b| > 0.000001 である間繰り返し、
  c ← (a + b) / 2。
  もし f(c) > 0 ならば、
    b ← c。
  そうでなければ、
    a ← c。
  枝分かれ終わり。
  // a と b を出力する。　…テストのため
繰り返し終わり。
a を出力する。
```

これを JavaScript にしたものは次の通り (実行例は図 6.1、細かい値は実行系により違うことがある):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html><head><title>sample</title></head><body>
<script type="text/javascript">
function f(x) { return x*x - 2.0; }
var a = 0.0;
var b = 2.0;
while(Math.abs(a - b) > 0.000001) {
  var c = (a + b) / 2.0;
  if(f(c) > 0) { b = c; } else { a = c; }
  document.writeln('[' + a + ',' + b + ']<br>');
}
document.writeln('answer = ' + a + '<br>');
</script>
</body></html>
```

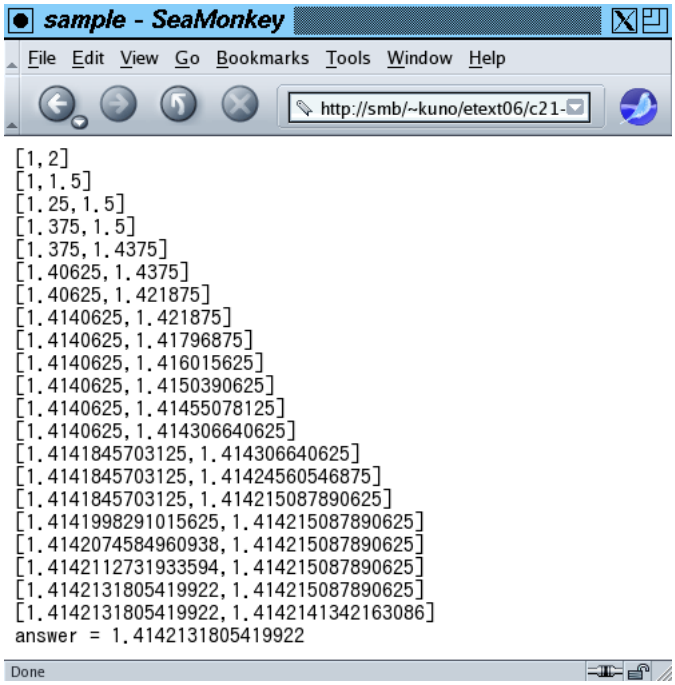


図 6.1: 2 の平方根を求める

誤差、終了条件、不変条件

先のプログラムの実行結果を見ると、確かに 2 の平方根が小数点以下 5 桁まで正しく求まっている。しかし逆に言えばそこから先は正しくないから、これでは正しい解ではない…のだろうか?

実は先のプログラムで「どれくらい正確に」根を求めるかは、while 文の条件にある値「`0.000001`」で指定されていた。だから、これをもっと小さな値にすればもっと精度のよい解が求められる。しかしいくらでもというわけではない。

もともと、コンピュータによる計算はある決まった有限の精度内で行われるので (JavaScript 処理系の場合は 16 桁程度)、それより正確な値は求められない。それでは不十分だろうか? もともと、$\sqrt{2}$ は無理数だから、何百桁求めてもその先は無限にあるわけで、それを書き下すことはあまり意味がない。むしろ、必要な精度を決めてその精度内で計算することの方が普通なわけである。

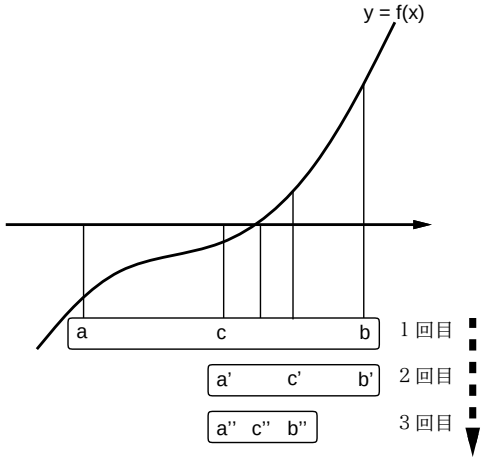


図 6.2: 区間 2 分法の原理

では次に、なぜこれで $f(x)$ の根が求まるかを考えてみる (図 6.2)。$f(a)$ が負、$f(b)$ が正なのだから、その間のどこかで $y = f(x)$ のグラフは X 軸を横切る。そこの x の値が根である。つまり根は区間 $[a, b]$ の中にある。

ここで、$c = \frac{a+b}{2}$ を計算して、$f(c)$ を求める。もしそれが正であれ

ば、根は c より右にあるはずである。また負であれば、根は c より左にあるはずである (図 6.2 の場合)。そのいずれかによって、a か b のどちらかを c で置き換えたとすると、新しい $[a', b']$ の区間の幅は前の半分になっていて、なおかつこの区間内に根があるという状況は同じである。

これを何回も繰り返すことで、区間の幅は半分ずつになるが、根がこの中にあることは変わらない。そこで、区間の幅が許される誤差以下になった時に終われば、その区間のどこの値でも求める根に十分近い値になっているわけだ。

この時、次の 2 つの性質に注意したい。

(a) 変わらない性質 — 根は区間 $[a, b]$ 内にあるということ。
(b) 変わって行く性質 — 区間の幅は毎回半分ずつになるということ。

繰り返しは区間の幅が一定以下になったら終わるのだから、(b) からこのアルゴリズムは必ず停止することが分かる。そして、(a) から停止したときに a や b は根に十分近いことが分かるわけである。このように、繰り返しを持つアルゴリズムでは「何が変わらないか」「何が変わって行くか」に注目して正しさをチェックするのがよい。

演習: 上のプログラムを手直しして、もっと正確な値を求めてみよ。また、3 の平方根、2 の 3 乗根なども求めてみよ。

6.1.3 再帰を含むアルゴリズム

関数と再帰呼び出し

関数は外部から呼び出されて値を計算し、結果を返すようなものだった。**再帰関数**とは、その計算の中で直接または間接に自分自身を呼び出すようなものをいう。

たとえば、階乗を考えてみよう。n の階乗 $n!$ は $1 \times 2 \times \cdots \times n$ だが、これを次のように書くこともできる:

- $0! = 1$。
- $n! = n \times (n-1)!$($n > 0$ のとき)。

これをそのまま、疑似コードや JavaScript のコードにすることができる:

```
factorial(n) : n の階乗を返す
  n = 0 ならば、
    1 を返す。
 そうでなければ、
    n × fact(n-1) を返す。
 枝分かれ終わり。
```

```
function fact(n) {
  if(n == 0) {
    return 1;
  } else {
    return n * fact(n-1);
  }
```

これは正しく動くのだろうか? fact の定義の中で fact を利用してしまったら「堂々めぐり」になるのではないだろうか?

実際には堂々めぐりにはならず、ちゃんと動く。たとえば fact(5) を計算するようすを見てみよう:

```
fact(5)
5 × fact(4)
     ↓
5 × 4 × fact(3)
         ↓
5 × 4 × 3 × fact(2)
             ↓
5 × 4 × 3 × 2 × fact(1)
                 ↓
5 × 4 × 3 × 2 × 1 × fact(0)
5 × 4 × 3 × 2 × 1 × 1 ←
5 × 4 × 3 × 2 × 1 ←
5 × 4 × 3 × 2 ←
5 × 4 × 6 ←
5 × 24 ←
120 ←
```

なぜ堂々めぐりにならないかというと、このコードが次のようになっているからである:

- 自分自身を呼び出すところでは、「元より 1 段簡単な問題」のために自分自身を呼ぶ (たとえば、5 の階乗を求めるとき、4 の階乗を求めるのに自分自身を呼ぶ)。

- 簡単な場合は別途扱う (たとえば 0 の階乗は 1 なので単に 1 を返す)。

言い替えれば、正しくこのような構造になっていれば、再帰的な関数では正しく計算できて値が求まるといえる。

例題: 最大公約数

もう少し込み入った例として、最大公約数を求める再帰関数を作ってみる。正の整数 x、y の最大公約数には次の性質があることに注意:

- $x = y$ なら、x と y の最大公約数は x 自身。
- $x > y$ なら、x と y の最大公約数は $x - y$ と y の最大公約数に等しい。
- $x < y$ なら、x と y の最大公約数は x と $y - x$ の最大公約数に等しい。

これをもとに、疑似コードを作ってみる:

```
gcd(x, y) : 2 つの正の整数 x, y の最大公約数:
  もし x = y なら、
    x を返す。
  そうでなくて x > y ならば、
    gcd(x-y, y) を返す。
  そうでなければ
    gcd(x, y-x) を返す。
  枝分かれ終わり。
```

たとえば 18 と 12 に対して上の定義を適用したようすを見てみよう:

```
gcd(18, 12)   : x が大きいので gcd2(x-y,x) を呼ぶ
→ gcd(6, 12) : y が大きいので gcd2(x,y-x) を呼ぶ
→ gcd(6, 6)  : x と y が等しいので x が答え
→ 6
```

JavaScript コードも次の通り:

```
function gcd2(x, y) {
  if(x == y)     { return x; }
  else if(x > y) { return gcd2(x-y, x); }
  else           { return gcd2(x, y-x); }
}
```

なお、この if 文の形は先に出て来た疑似コードの「そうでなくて〜なら」に対応するものだが、ある if 文の else 文にすぐ次の if 文がつながったもので、**if-else の連鎖**と呼ばれる。if-else の連鎖は、一般に次のような形になる。

```
if(条件 1) {
  動作 1
} else if(条件 2) {
  動作 2
} else if(条件 3) {
  動作 3
…
…
} else {
  動作 X
}
```

if-else の連鎖は、上から順番に「条件 1」「条件 2」…を調べて行き、最初に「真」だった条件に対応する動作を実行し、どれも真でなければ「動作 X」を実行する形になっていて、理解しやすい。

演習: 再帰アルゴリズム/再帰関数を使って、正の整数 n を 2 進数の文字列に変換せよ。

6.2　データ構造とアルゴリズム

6.2.1　配列を用いるアルゴリズム

データ構造、配列

ここまでに学んで来たアルゴリズムは、たかだか 2、3 個の変数に保持している数値を扱うものだった。しかし当然ながら、世の中には、もっと多くのデータを扱うアルゴリズムやプログラムが多数ある。

このとき、プログラムが扱うデータをどのような形 (順番や配置) で保持しておくかということが、それを扱うアルゴリズムのありかた (書きやすさや効率) にも大きな影響を与える。この、プログラムが扱うデータの順番や配置などを一般に**データ構造**と呼ぶ。

データ構造の中でももっともよく使われるのは、「**並び**」すなわち一連の値が順番に並んだ配置である。多くのプログラミング言語は、値の並びのデータ構造を扱うために、**配列**と呼ばれる機能を備えている。

配列とは、数学で言えば**添字**つきの変数 (x_0、x_1、…) に相当するもので、次のような性質を持っている。

- 概念的には、同じ種類の値が並んだものである。
- 配列の名前と「何番目か」という指定 (添字) を組み合わせて個々の要素を指定し、読み書きできる。
- 何番目の要素でも、同じ手間で読み書きできる。

JavaScript を含む多くの言語では、並んだ値が何番目のものかを表す指定を `[ ]` の中に書いて、たとえば `a` が配列だとすると、`a[0]`、`a[1]` などのようにあらわす。[1]この番号をつけた一つひとつの要素は、変数と同じように値を代入したり、そこから値を取り出したりできる。

上の例のように、コンピュータでは番号は 0 からつけることが多い。したがって、要素数 (並んだ値の個数) が 10 であれば、`a[0]`、`a[1]`、…、`a[9]` までの要素を使うことができる。

また、JavaScript では、配列もオブジェクトであり、要素数は length というプロパティに格納されている (つまり、配列 a の要素数は a.length で参照できる)。一般に、配列 a で使える添字の範囲は 0 から a.length-1 までということになる。

JavaScript で配列を作り出す方法はいくつもあるが、代表的なものを説明しておく。

- `a = [4, 3, 1, 2];` — 角かっこの中に値を「`,`」で区切って並べると、並べた値をこの順に格納した要素を持つ配列が作り出される。
- `a = new Array(5);` — 「`new Array(個数)`」という書き方で、指定した個数の要素を持つ配列が作り出される。この時は、すべての要素は、null という、「まだ何も入っていない」ことを表す特別な値が格納された状態になっている。上の例は「`a = [null, null, null, null, null];`」と同じことになる。

[1]言語によっては書き方が違うものもあるが、C、C++、Java、JavaScript などはすべてこの流儀である。

- a = 'This is a pen'.split(' '); — 文字列の split() メソッドは、文字列を指定した文字のところで切り分けて配列として返す機能を提供する。ここでは空白のところで切り分けているので、上の例は「a = ['This', 'is', 'a', 'pen'];」と同じことになる。

例題: 合計と平均

例として、数値の並びを配列に読み込み、その合計と平均を求めるアルゴリズム考えよう。まず大まかな疑似コードを示す。

```
{数値の並びを配列 a に読み込む。}
{変数 total に合計を求める。}
{変数 average に平均を求める。}
total と average を出力する。
```

数値の並びを読み込むには、文字列を読み込んで split() で分けて配列にし、それぞれの要素を数値に変換する。この部分はひとまとまりの機能なので、手続きとして考えよう:

```
readNumbers(m) : メッセージ m を表示し数値の配列を読み込む:
  a ← メッセージ m を表示して文字列を読み込み配列に分解。
  i を 0 から a.length の手前まで増やしながら繰り返し、
    a[i] ← a[i] を数値にしたもの。
  繰り返しおわり。
  a を返す。
```

「ある値の手前まで」の繰り返しには、for 文で条件として「<」を使えばよいことに注意。この JavaScript コードを示す:

```
function readNumbers(m) {
  var a = prompt(m).split(' ');
  for(var i = 0; i < a.length; ++i) {
    a[i] = parseFloat(a[i]);
  }
  return a;
}
```

合計を求めるには、変数 total を最初 0 にしておき、配列の要素を順に「足し込んで」行けばよい。平均は合計を要素数で割れば求まる。これらをそれぞれ具体的にしたものを次に示す。

```
a ← 数値の配列を読み込む。
i を 0 から a.length の手前まで 1 ずつ増やしながら繰り返し、
  total ← total + a[i]。
繰り返しおわり。
average ← total / a.length。
total と average を出力する。
```

具体的になったので、この疑似コードを JavaScript に直してみる。

```
var a = readNumbers(' 数を空白で区切って入力してください。');
var total = 0;
for(var i = 0; i < a.length; i = i + 1) {
  total = total + a[i];
}
average = total / a.length;
document.write(a + ' の合計は' + total +
               '、平均は' + average + ' です。<br>');
```

演習: 配列を使って次のようなプログラムを作りなさい。

- 数値の並びを読み込み、最大値と最小値を出力する。
- 2 つの数値の並びを読み込み、平均値が大きい方を出力する。

6.2.2 データ構造の工夫

例題: 素数の列挙

先に述べたように、データ構造はアルゴリズムの書きやすさや効率に大きく影響する。言い替えれば、データ構造を工夫することで、込み入ったアルゴリズムを簡単なものにしたり、時間の掛かるアルゴリズムを速くしたりできることがある。

例として、与えられた整数 N 未満の素数をすべて出力するアルゴリズムを考えてみる。アルゴリズムとして、素数の候補とする数 k を 2 から順に用意して、それが素数であるかどうか調べ、素数だったら打ち出すという方針のものを作ってみる。

素数かどうかの素朴な判定方法としてとりあえず、2 から k-1 までの数で割ってみて割り切れるかどうか調べるアルゴリズムを用いることとして、そのための関数を用意する:

```
sosu(n) : n が素数かどうかを判定
  i を 2 から n の手前まで変化させながら繰り返し、
    もし n % i == 0 なら、
      「いいえ」を返す。
    枝分かれ終わり。
  繰り返し終わり。
  「はい」を返す。
```

```
function sosu(n) {
  for(var i = 2; i < n; i = i + 1) {
    if(n % i == 0) { return false; }
  }
  return true;
}
```

なお、true と false は「はい」「いいえ」を表す JavaScript の値であり、if 文などの条件にそのまま書くことができる。

次に、プログラム全体の疑似コードおよび JavaScript コードは次のようになる:

```
整数 n を読み込む。
k を 2 から n の手前まで 1 ずつ増やしながら繰り返し、
 もし sosu(k) ならば、k を出力する。
繰り返しおわり。
```

```
var n = parseInt(prompt(' いくつ未満の素数を表示しますか。'));
for(var k = 2; k < n; k = k + 1) {
  if(sosu(k)) { document.write(k + ' '); }
}
```

演習 このプログラムを動かして、5 秒間でいくつくらいまでの素数が表示できるか調べておこう。

データ構造の追加による効率化

上のアルゴリズム別の、次のような考え方について見てみよう (図 6.3)。

- 0〜$n-1$ の数が並んでいて、必要に応じて印をつけるものと考える。
- 2 から順に素数を調べていく。
- 2 は印がついていないので素数。2 の倍数すべてに印をつける。
- 次の印がついてない数は 3 なので 3 は素数。3 の倍数すべてに印をつける。
- 次の印がついてない数は 5 なので 5 は素数。5 の倍数すべてに印をつける。
- …

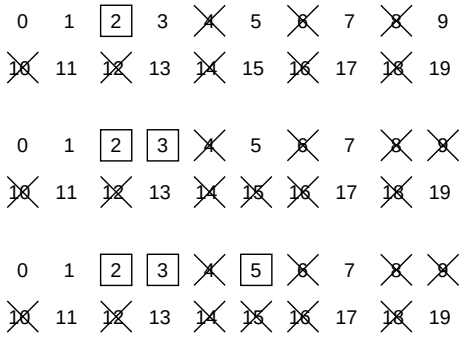


図 6.3: エラストテネスのふるいの原理

これを実装する場合には、配列を使って、その i 番目に何かを入れることで「数 i に印をつける」ことを表すことができる。この考え方が目新しいのは、配列を何らかのデータを入れるのに使うのではなく、「i 番目に印がついているかどうか」という情報のために使うところである。では、この考え方によるアルゴリズムを示そう。

```
正の整数 n を読み込む。
添字の範囲が 0〜n の配列 a を用意する。
x を 2 から n の手前まで変化させながら繰り返し、
  もし a[x] に印がついていなければ、
    x を出力する。 // x は印がないので素数
    i を x から n の手前まで x ずつ変化させながら繰り返し、
      a[i] に印をつける。
    繰り返し終わり。
  枝分かれ終わり。
繰り返し終わり。
```

JavaScript では、最初に大きさを配列を作った時はすべて null が入っていることを利用して、null 以外の値、たとえば「1」を入れることで「印」をつけることができる。この方針で JavaScript プログラムを作ってみた:

```
n = parseFloat(prompt(' 正の整数 n を入力'));
a = new Array(n);
for(var x = 2; x < n; ++x) {
  if(a[x] == null) {
    document.write(' ' + x);
    for(var i = x; i < n; i = i + x) { a[i] = 1; }
  }
}
```

このアルゴリズムの方が、先のアルゴリズムよりも簡潔で読みやすい (先のアルゴリズムは関数に分けないと大変分かりにくくなる)。また、このアルゴリズムの方が実行速度もずっと速い。このように、データ構造を工夫することで、分かりやすい/速いアルゴリズムを作ることができる場合がある。

演習 こちらのアルゴリズムでは 5 秒間でいくつ未満の素数を列挙できるか調べ、先のアルゴリズムと比較してみよ。

6.3 実際の問題への適用

6.3.1 レコード

レコードとその用途

前節で学んだ配列は、複数のデータを含むデータ構造であり、次のような性質を持っていた:

- 同種類のデータが並んでいる
- その「何番目」という番号を指定して参照できる

これに対し、複数のデータを含むデータ構造のもう 1 つの種類として、**レコード**と呼ばれるものがあり、それは次のような性質を持つ:

- 必ずしも同じでない種類のデータが並んでいる
- 各データには「名前」があり、その名前で参照できる

なお、レコードに含まれる個々のデータの格納場所のことを**フィールド**と呼ぶ。

たとえば、本のデータを扱うことを考える。1 つの本には「題名」「著者」「価格」などのデータが付属しているので、これらのデータを組み合わせたレコードで 1 つの本を扱うのが自然である。

ここでは JavaScript で使う書き方に合わせて、1 つのレコードのデータを次のような形で書き表すことにする:

```
{ 名前:値, 名前:値, … }
```

たとえば本の場合なら次のような具合である。

```
{ dai:'日本の天気', cho:'山田太郎', nedan:1200 }
```

JavaScript では、レコード値も他の値と同様に変数に入れて扱うことができる:

```
var book = { dai:'日本の天気', cho:'山田太
郎', nedan:1200 };
```

ここで、変数 book に格納されているレコードの各フィールドは、変数名の後ろに「. フィールド名」をつけることで参照できる:

```
document.write(book.dai + '<br>');  // 題名を表示
book.nedan = book.nedan - 100; // 値段を 100 円引き
```

この「.」は日本語の「の」だと考えると分かりやすい。つまり上の例では「本の題名」や「本の値段」を参照していたわけである。

レコードの配列

1 冊の本の情報であれば上記のようにして 1 つのレコードで表せるが、実際に我々の身のまわりにある情報をコンピュータで扱う場合は、たとえば「持っている本全体の価格の合計を求める」などのように、多数のもののデータをまとめて扱うのが普通である。そのような場合には、**レコードの配列**、すなわちレコード値を沢山並べた配列という形で扱うのが一般的である。

たとえば JavaScript で自分が持っている本のデータを配列 books に入れるとしたら、次のように「[…]」の中にレコードデータを並べて書いたものを books に入れるようにすればよい:

```
var books = [
  { dai:'日本の天気', cho:'山田太郎', nedan:1200 },
  { dai:'天気図', cho:'川名次郎', nedan:1300 },
  { dai:'高層天気図', cho:'中川大作', nedan:1800 },
  { dai:'大気の仕組み', cho:'山本健', nedan:2200 },
  { dai:'雲のはなし', cho:'山田太郎', nedan:1600 } ];
```

続いて、これらの本の値段の合計を求めるとしたら、前にやった方法と同様にして、ただしこんどは nedan フィールドについての合計を求めるようにすればよい:

```
var total = 0;
for(var i = 0; i < books.length; i = i + 1) {
  total = total + books[i].nedan;
}
document.writeln('本の価格合計 = ' + total + '<br>');
```

なお、実際に使われるプログラムでは、データはもっと沢山あり、プログラムとは別の場所 (ファイルやデータベース) に保存されていて、プログラムそれを読み込んで動くように作ってあるのが普通である。ここでは簡単に練習してみるために、プログラムの先頭にデータも一緒に書く形にしてある。

6.3.2 探索

線形探索のアルゴリズム

探索とは、配列などに格納されている複数のデータの中から求めるものを見つけ出すことをいう。たとえば、上の本のデータの中で「指定された題名のものを見つける」ことは探索に相当する。このとき、見つけるものを指定するデータのことを**探索キー**という。前記の例では「題名」が探索キーということになる。

探索のアルゴリズムにはさまざまなものがあるが、ここでは一番簡単な**線形探索**を見てみることにする。線形探索のアルゴリズムは次のようになる:

```
linearsearch(key) --- 配列に対する線形探索
  i を 0 から配列の上限の手前まで変えながら繰り返し
    もし{i 番目の要素が key と一致する}なら、
       i 番目の要素を返す。
    枝分かれおわり。
    見つからないという印を返す。
  繰り返しおわり。
```

線形探索を用いたプログラム

上では条件を一般的な形で書いてあるが、それを本の題名の場合について詳細化して JavaScript で書くと「`books[i].dai == key`」とすればよい。「見つからないという印」は、JavaScript では null を返すようにするのが普通である。

では、書名を入力してその情報を打ち出す JavaScript プログラムを示す:

```
var books = [
  { dai:'日本の天気', cho:'山田太郎', nedan:1200 },
  { dai:'天気図', cho:'川名次郎', nedan:1300 },
  { dai:'高層天気図', cho:'中川大作', nedan:1800 },
  { dai:'大気の仕組み', cho:'山本健', nedan:2200 },
  { dai:'雲のはなし', cho:'山田太郎', nedan:1600 } ];
var title = prompt('探す本の題名は? ');
var book = linearsearch(title);
if(book == null) {
  document.write('見つかりませんでした。');
} else {
  document.write('題名:' + book.dai + ' 著者:' +
                 book.cho + ' 価格:' + book.nedan);
}
function linearsearch(key) {
  for(var i = 0; i < books.length; i = i + 1) {
    if(books[i].dai == key) { return books[i]; }
  }
  return null;
}
```

6.3.3　整列

単純選択法のアルゴリズム

整列とは、データを「特定項目の大きい (または小さい) 順」に並べることを言う。このとき、整列の基準に使う項目のことを**整列キー**と呼ぶ。たとえば、値段の高い順に並べるのであれば、整列キーは値段ということになる。

整列のアルゴリズムにはさまざまなものがあるが、ここでは一番簡単なものの 1 つである**単純選択法**を見てみることにする。単純選択法のアルゴリズムは次のようになる (大きい順に並べる場合):

```
selectionsort() : 単純選択法で配列を昇順に整列
  i を 0 から配列の上限の手前まで変えながら繰り返し
    j ← {i+1 から配列の末尾までの間でキーが最大の項目番号}
    {i 番目と j 番目の項目を入れ替える}
  繰り返し終わり
```

なぜこれでよいかというと、まず繰り返しの 1 周目で配列の 0 番目に最大の要素が置かれ、2 周目で配列の 1 番目に残りのうち最大の要素が置かれ、…のように進んで行くから。

指定範囲のうちで最大の項目番号を調べるのには次のようにすればよい:

```
select(a, b)
    --- 配列の a から b の手前までで最大の項目番号を返す
  max ← 配列の a 番目
  k を a+1 から b の手前まで変えながら繰り返し、
    もし k 番目のキーが max のキーより大きいなら、
      max ← 配列の k 番目
    枝分かれ終わり。
  繰り返し終わり
  max を返す。
```

単純選択法のプログラム

i 番目と j 番目の交換についてはさほど複雑でないので関数を分けることはしない。全体を JavaScript に直したものを示す:

```
var books = [
  { dai:'日本の天気', cho:'山田太郎', nedan:1200 },
  { dai:'天気図', cho:'川名次郎', nedan:1300 },
  { dai:'高層天気図', cho:'中川大作', nedan:1800 },
  { dai:'大気の仕組み', cho:'山本健', nedan:2200 },
  { dai:'雲のはなし', cho:'山田太郎', nedan:1600 } ];
function selectionsort() {
  for(var i = 0; i < books.length; i = i + 1) {
    var j = select(i, books.length);
    var x = books[i]; books[i] = books[j]; books[j] = x;
  }
}
function select(a, b) {
  var max = books[a], maxpos = a;
  for(var k = a+1; k < b; k = k + 1) {
    if(books[k].nedan > max.nedan) {
      max = books[k]; maxpos = k;
    }
  }
  return maxpos;
}
selectionsort();
for(var i = 0; i < books.length; i = i + 1) {
  document.write('題名:' + books[i].dai + ' 著者:' +
    books[i].cho + ' 価格:' + books[i].nedan);
}
```

第7章　メディアリテラシーと情報倫理

7.1　メディアリテラシー

7.1.1　新しいメディアの出現

メディアとコミュニケーション

私たちは自分の考えや意見など、情報を誰かに伝達したり、入手する為の手段として、**コミュニケーション**を行っている。コミュニケーションの手段はさまざまである。直接会話を交わすことはコミュニケーションの主要な手段だが、それ以外にも次のような場合が考えられる (一例として):

- 時間的距離がある場合 — 伝言板
- 場所的距離がある場合 — 電話
- 時間的、場所的の両方の距離がある場合 — 手紙

このような、コミュニケーションにおける手段・媒体のことを**メディア**と呼ぶ。会話、手紙などはその代表的なものである。メディアを上手に利用することで、遠く離れた人ともコミュニケーションをとることが可能である。また、テレビやラジオ、雑誌や新聞など、一方的に多数の人に情報を伝えるメディアを**マスメディア**と呼ぶ。

情報化社会と呼ばれる今日では、新たに情報通信ネットワークを用いた電子メールや Web 掲示板などの出現により、従来のメディア (情報通信ネットワークをも用いないメディア) に比べ時間的にも場所的にもその距離が限りなくゼロに近づいてきた。

例えば、電子メールと手紙を比べると、文章を遠く離れた相手に届ける点では同じだが、届くまでの時間が短縮されている。また電話とテレ

ビ電話を比べると、後者ではディスプレイに相手の顔が映るため、目の前にその人が居るような感覚で会話をすることができる。

ネットワーク上の様々なメディア

具体的に情報通信ネットワークを用いたメディアにはどのようなものがあるのだろうか。以下に主なものを紹介する:

電子メール: インターネットを利用して、文字情報の送信・受信を行うことが出来る。また、データファイル等を添付して送ることも出来る。電子メールはいつでも手軽に送信することが出来るが、いつ相手がそのメールを読むか分からないため、緊急性のある内容のやり取りには適さない。また、大容量のデータを送信すると、ネットワークが混み合い他の利用者の迷惑となる。そのため、動画などの配信には適さないし、やむなく送信しなければならない場合は、極力データサイズを少なくしてから送信する心遣いが必要だ。

メーリングリスト: 特定のグループ内での情報交換に用いる電子メールの利用法の一つ。特定のメールアドレスにメンバーを登録することで、そのアドレスに届いたメールをメンバー全員に転送するシステム。登録メンバー全員に同じメールが行くため、グループメンバーによっては個人情報など気を使う必要がある。

Web 掲示板: Web ページ形式の一つ。さまざまな話題において、誰でも自由に意見を書くことが出来るページの事。書き込んだ意見は修正することは出来ないが、。嘘の情報を面白おかしく書き込む人もいるため、情報の信憑性に乏しい。国内の代表的なものには「2ちゃんねる http://www2.2ch.net/ 」がある。

ブログ: Web ページ形式の一つ。Web Log の略で、Web 上に残される Log(記録) という意味をもつ。主に日記など時系列で比較的頻繁に更新される Web ページのこと。ブログは単に日記としての行動記録に留まらず、日々の生活で感じた事柄や時事ニュースへの見解などを掲載するのが主流となっている。また、日記に対するコメントを書き込む機能もあり、著者と読者間での意見交換をすることができる。

Wiki: Web ページ形式の一つ。誰でも自由に編集できる Web サイトの事。Web ブラウザがあればどこからでも利用できるため、Web 上のメモ帳や、グループでの共同作業の場として利用することができる。国内の代表的なものには「ウィキペディア http://ja.wikipedia.org/」がある。

SNS: Web 上でのサービスの一つ。Social Networking Service の略で、人と人との繋がりを電子的にサポートするサービスのことである。検索機能を用いることで、共通の話題の相手を見つけることができ、新たな友人の輪を広げることができる。参加方法は、単にメールアドレスを登録すればよい場合と、会員からの招待が無ければ参加できない場合の 2 つのシステムがある。後者の場合はインターネットにおける匿名性の問題をある程度改善されている。国内の代表的なものには「mixi(ミクシィ) http://mixi.jp/home.pl 」がある。

出会い系サイト: Web 上でのサービスの一つ。不特定の男女が出会いを求める目的で運営されている Web サイトのこと。Web サイト閲覧機能を持った携帯電話の登場により、ユーザー年齢が低年齢化し、未成年ユーザーの流入が起きたことにより、援助交際や売春に加え殺人事件などのさまざまな犯罪が起き、2003 年には 18 歳未満の人を誘い込む行為などを禁止する「出会い系サイト規制法」が制定されている。

P2P: ネットワークの形態の一つで、通常は処理を依頼する側 (クライアント) と、依頼を実行する側 (サーバ) があり、私たちが一般的に利用するのはクライアント側である。しかし、P2P は特定のクライアント、サーバを持たずに、お互いがクライアントとしてもサーバとしても動くような仕組みになっている。サーバを別途用意する必要が無いため、低コストで実現することができる。また、WinMX や Winny などに代表されるファイル共有ソフトにも P2P は利用されている。ファイル共有ソフトとは、インターネットに繋がった匿名のコンピュータ間で自由にファイルの交換が出来る。そのため、著作権を侵害した利用が行われることもしばしばある。

演習: 情報通信ネットワークを利用するメディアと従来のメディアを挙げて、表にしてみよう。

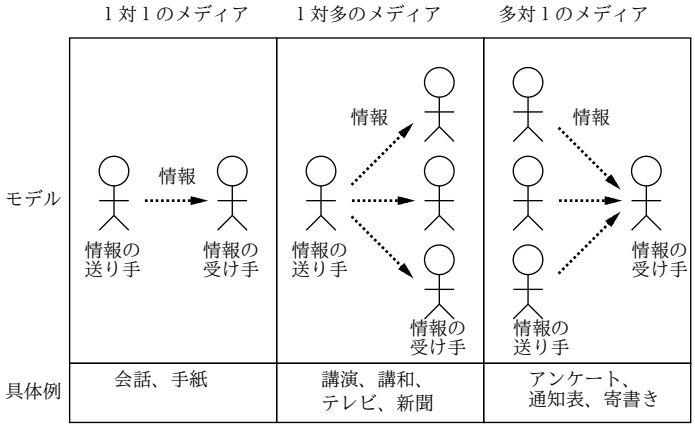


図 7.1: 送り手と受け手の数によるメディアの分類

7.1.2　メディアの分類と比較

メディアの分類

私たちはコミュニケーションを用いて情報の伝達を行っている。情報の伝達の手段 (メディア) はさまざまだが、それには必ず送り手と受け手が必要である。その送り手と受け手が単体か複数かによってメディアを分類することができる。また、情報がどのように手元に来るのか、能動的か受動的かによって pull の型か push の型に分類することができる。

送り手と受け手の数による分類には、次のようなものがある:

1 対 1 のメディア: 送り手も受け手も単体の場合。直接会話や電話、手紙など。

1 対多のメディア: 送り手は単体であるが相手は複数の場合。授業や講演会などの講話、テレビや新聞など。

多対 1 のメディア: 送り手は複数あるが相手は単体の場合。アンケートや寄書き、通知表など。

能動的か受動的かによる分類内容には、次のものがある:

pull 型のメディア: 知りたい情報をリクエストし、それに対して答えが返ってくるようなメディア。会話や Web ページなど。(Web ペー

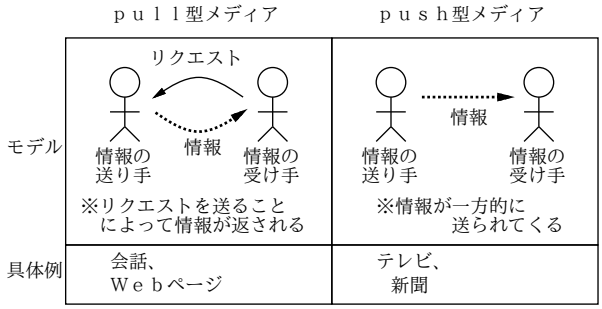


図 7.2: 能動/受動よるメディアの分類

ジはサーバに対して依頼を出し、情報を取りに行きその結果を表示させるもの)

push 型のメディア: 一方的に情報が送られてくるメディア。受け手はその中から情報を選らんで取得する (受け手が欲しいと思った情報とは必ずしも一致するとは限らない)。テレビや新聞など。

図 7.1、図 7.2 に、これらのメディア分類のモデルと具体例を挙げる。

演習: 前項の課題で挙げた従来のメディアと情報通信ネットワークを用いたメディアを分類し、図表に記入してみよう。

様々なメディアの比較

図 7.1、図 7.2 を見てみると、全ての項目に対して、従来のメディアと情報通信ネットワークを用いたメディアの双方ともに当てはまる事が確認できる。これは情報通信ネットワークを用いたメディアは、従来のメディアに応用 (情報通信技術) を加えたものであり、基本的な部分 (文字を伝える、音声を伝える、絵 (画像・動画) を伝えるなど) は従来のメディアと同じだからである。しかし、前項で紹介したように、情報通信ネットワークを用いたメディアは従来のメディアに比べて時間的・場所的距離がゼロに近づいてきており、気軽に使えるようになっている。では、基本的な部分は同じで時間も場所も気にせず使えるならば、情報通信ネットワークを用いたメディアは従来のメディアに全て取って代わるものなのだろうか。

そこで、日常的に利用される 1 対 1 のメディア、1 対多のメディアに関して、従来のメディアと情報通信ネットワークを用いたメディアにおける、利用する際のメリット・デメリットを比較してみよう。

まず、1 対 1 のメディア、手紙や葉書と電子メールについて比較してみる。双方とも文字を相手に伝えるメディアである。まずコストの面では手紙や葉書には切手代など輸送代金がかかってしまう。しかし、電子メールは基本的に無料で利用することが出来る。また、相手に届くまでの時間は圧倒的に電子メールの方が早い。ならば手紙などは不要で電子メールのみあれば良いのだろうか。

例えば、電子メールに書かれる文字の見た目は無機質な感じがし、誰が書いても変わらない字体をしている。一方手紙や葉書に書かれる文字は、送り手の癖や気持ちなどが字体に表れるため、より相手を感じることが出来る。このように捉えると、文通などで次の手紙が届くまでの時間も愛おしく意味のある時間だと感じることが出来る。

手紙、電子メールなどを利用する際に注意しなければならないのは、送信しても相手が必ずしもそれを読んでいるとは限らないということである。特に電子メールではウィルスやスパム等の有害なメールと間違われて読む前に削除されてしまう可能性もある。そのため、送信したからと言って共通認識だと捉えてしまうのは良くない。

次に、1 対多のメディア、新聞や図書と Web 上の新聞について比較してみる。双方とも文字、写真等を用いて過去の情報を伝えるメディアである。まずコストの面では、新聞や図書を読むためには当然代金がかかるが、Web 上の新聞は基本的に無料である。また、新聞に掲載されている出来事は基本的に昨日の事である。しかし、Web 上の新聞であればリアルタイムに更新することが可能であるため、常に最新の情報が掲載されている。また、リンク機能を利用すれば関連記事などの参照も容易い。ならば、新聞は不要で Web 上の新聞だけがあれば良いのだろうか。

例えば、「10 年前の今日起きたことを調べよう」と思ったとき、Web 上では 10 年も前のことは調べることは難しい。しかし、新聞であれば縮小版のバックナンバーが図書館等に保存されており参照することができる。また、過去の情報を調べるには 1 日の記事である新聞よりも製本された図書を用いたほうがより詳しい内容を知ることができる。このように、過去の情報を伝えるメディアであっても、知りたい情報によりメディアを選択しなければ適切な情報は得られないのである。

このように、従来のメディアと情報通信ネットワークを用いたメディ

アにはそれぞれ優れている部分があることが分かる。そのため、これから行おうとするコミュニケーションにおいて、どのメディアが適切なのかを判断して利用する必要があることが分かる。

一般に、さまざまなメディアが持つ特性や制約を知って、場面に応じてそれに適したメディアを適切な形で活用する能力を**メディアリテラシー**という。情報社会の今日においては、すべての人が適切な水準のメディアリテラシーを身につけることが望まれている。

7.2 情報倫理と安全性

7.2.1 情報倫理とその考え方

倫理と情報倫理

あなたが子供の親であり、自分の子供に「人のものを盗ったらいけません」と言い聞かせたとき「なんで?」と質問し返されたら、何と答えるだろうか?

その答えとして、「法律で禁止されているから」というのはサイテーである。なぜなら、その次は「法律で禁止されていなければやってもいいの?」になるからである。

この「いい/悪い」の判断を (法律など) 他人に押しつけていては、人間としての主体性がなくなってしまう。そうではなく、自分の中で主体的に「何は正しく、何は正しくないか」を判断する基準や方法を持つことが、特に価値観の多様化する現代においては重要である。**倫理学** (Ethics) とは、哲学の一分野であり、「何は正しい/正しくない」を考える学問分野である。

倫理学自体は長い歴史を持ち、また倫理学という名称は使っていなくても、我々は道徳の時間や実体験を通じて「何は正しい/正しくない」の問題についても多くのことを学んで来たはずである。

しかし、今日のコンピュータとネットワークの発達は、人と人の関わりにおいて、これまでになかったさまざまな新しい特性の場面を生み出している。そのような新しい場面では、これまでに身につけてきた「正しい/ 正しくない」の考え方をどのように適用したらいいかが、多くの人にはまだ十分分かっていないという状況がある。そして、このような

新しい状況での「正しい/正しくない」の考え方ないし枠組みのことを**情報倫理**と呼ぶ。

定義 情報倫理とは、コンピュータやネットワークなどの新しい情報技術によって生まれた新しい場面に、倫理的考え方をあてはめることを言う。

情報倫理の土台となる前提

では具体的に、ネット社会における「正しい/正しくない」をどのように考えたらいいのだろうか。この問いについて、次の前提を置くことにする:

- ネット上の場でも、正しい/正しくないの基準は現実社会と同じである。
- 他人の嫌がることはやってはいけない。
- 他人と意思疎通し、他人の立場に立って考えることは価値がある。

最初の項目については、ネット上の場が「特別である」という意識を否定している。実際、ネットワークの初期においては「インターネット上には法律は存在しない」というとんでもない説を真面目に唱える人もいたが、ネット上の詐欺や不法行為によって逮捕されたり損害賠償を請求されることも珍しくない今日では、ネットも現実社会の中に「埋め込まれた」ものであり、同じ規則が適用されるということは広く知られるようになっている。

2 番目の項目については、「**情報モラル**」「**ネチケット**」などの言葉で表されることもあるが、実は 1 番目の項目の帰結でもある。つまり、現実世界でも「他人の嫌がることはやってはいけない」は当然の規則として受け入れられている。しかしなぜだろうか? これは、モラルに関する広く受け入れられた考え方である**黄金律** (自分がされたくないことは、他人にもしてはいけない、なぜなら他人が自分にそれをしたら嫌だから) から来ている。[1]

3 番目の項目はやや間接的であるが、2 番目の項目の前提とも言える。というのは、まず他人の立場に立とうという意思がない人には他人が

[1] 黄金律の前提として「人はみな平等」があることに注意したい。かっては身分制度や奴隷などに見られるように人は平等とは限らなかった。また、狂心的な犯罪者には「自分は他人と違って特別だからこれをやってもよい」という言動が見られることがある。

嫌がることが分からないので、他人が嫌がることを避けることもできない。さらに、嗜好は人によって様々なので、ある人が何を好ましいと思うか (逆に何は嫌だと思うか) は、ただ想像するだけでは不十分であり、きちんとコミュニケーションを取らない限り分からない。もっとも、普段から交流のある相手であれば、ことさら聞かなくても相手の嗜好が大体分かるということはあるだろうし、これまでの社会であればそのような「近い知合い」との交流がほとんどの場面を占めていたからそれで済んでいたかも知れない。しかし、今日のように発達した情報技術を通じて、会ったこともない非常に広い範囲の人と接する機会が増えたことによって、相手について「知っている」ことは少なくなり、きちんとコミュニケーションを取ることが重要になってきたといえる。

情報倫理の考えの適用例

上で挙げた原理がどのような形で適用されるのか、具体的な例にあてはめて考えてみよう。たとえば、次のような「やってはいけない」ことはなぜ「いけない」のだろうか:

> メールの添付ファイルで大きなデータをいきなり送ってはいけない。

ここで「添付ファイル」については次のことが考えられる。

- 普通のテキストとちがって、大きなデータになりやすい。
- テキスト以外の種別なので、それを扱うアプリケーションソフトが必要である。

また「大きなデータ」については次のことが考えられる。

- 携帯端末などでは大きなファイルの受信が制限される。
- ネット接続が速ければすぐ送れても低速だと時間が掛かる。[2]

次に「相手の立場に立って見る」と、次のことは好ましくないだろう:

- メールの受信に非常に時間が掛かる。
- 受け取ったファイルが読めない。

[2]たとえば 10MB のファイルだと、12Mbps の接続だと 1 秒程度で送れるが、56Kbps の音声モデムだと 2000 秒 (30 分以上) を要することになる。

逆に、これらの問題がないなら、つまり「相手のメール受信時のネット接続環境が十分高速であり」「相手が添付ファイルのデータを扱うアプリケーションソフトを使える状況なら」(つまり打合せる等してこれらの条件が確認できていれば) ファイルを送ってもよい。だから最初の文の「いきなり」は「条件を確認していない」という意味だったわけである。

このように、さまざまな「注意すべきこと」にはそれぞれ前提や根拠があり、それらをきちんと分かっていれば盲目的に規則に従わなくても自分で適否が判断できるようになる。そして、そのためには次の 2 つが必要なわけである:

- 関連する技術的/社会的なことがら (上の例では添付ファイルの性質や伝送速度のことがら) を知っている。
- 「土台となる前提」のあてはめ方を知っている。

たとえば上のような規則が沢山掛かれた本があって、それを暗記したとしても、情報技術の進歩は急速なので、翌年になればまた新しい「規則になっていない事柄」が現れるので、あまり意味がない。これに対し、原則からあてはめることができれば、新しいことがらについても (技術的ことがらはそれぞれ調べるとして)、そのつど的確に判断ができるようになるはずである。

情報倫理と法律・規則

冒頭で短絡的に法律を参照することは望ましくないと書いたが、では法律やその他の規則は考慮しなくてよいのだろうか? もちろん、そんなことはない。

法律とは、我々の社会において皆が共通に合意すること (の一部分) を規則化したものであり、従って個々の条文等が定めるところが自分の倫理的判断基準から導けるものでないとしても、「皆が合意して決めた決まりに従う」という倫理的判断から「法律を守る」ことが導かれる (わが国の法律は間接民主制に基づき国民の代表が制定している)。

では、法律や条例以外の規則、たとえば個々のサイトや企業等が設けている規則についてはどうだろうか。これらは国民 (や地方議会) の代表が決めているわけではなく、それぞれの管理者や企業が「勝手に」決めたものに過ぎない。

しかし、これらはその規則の及ぶ範囲に加わること (サイトを利用する、その企業のサービスを選択する等) は強制ではなく、それぞれが自分の意思で選択している。従って、選択時にこれらの規則が提示され、それを承諾して加わったのであれば、その規則を守ることは「自分が約束したことを守る」という倫理的判断から導かれることになる。[3]

なお、法律や条例でもその他の規則でも、その内容が自分の倫理的判断と相違する場合も多いにあり得る。そのような場合でも上のような考えに立つなら、「規則を守らない」ではなく「規則を手直しするべく働きかける」ことを選ぶのが倫理的原則にかなっていると言える。

7.2.2 情報社会と安全性

安全性の定義と位置づけ

セキュリティ (security、**安全性**) とは「危険や被害から護られている状態」を表す概念である。同じ「安全性」を意味する言葉として**セーフティ** (safety) もあるが、セキュリティの方が「外部の物や人間に由来する危険や被害から護る」という意味合いが強くなっている。

情報システムなどのシステムにおけるセキュリティの定義を次に示しておく:

定義 セキュリティとは、システムがユーザの意図したように使えること、およびシステムがユーザの意図しないことを行わないことを意味する。

この定義の前半では、たとえばユーザが使おうとしたときにシステムが妨害や欠陥のため思ったように動作しない場合、セキュリテイが低いということを述べている。また後半では、たとえばシステムがユーザの公開したくない情報を漏洩させてしまったり侵入者に勝手に使われてしまったりする場合、セキュリティが低いということを述べている。

情報システムは一般に、ユーザまでを含めた全体としてシステムが存在しているので、システム自体が適切に設計され構築されていても、ユーザであるあなたの行動が適切でなければ、システム全体のセキュリ

[3] 参加時にきちんと規則について説明しないサイトや企業は倫理的でない行動を取っていると言える。なお、一定形態での商品やサービスの販売に当たっては、予め規則を明示するよう定めた法律がある。

ティも低下してしまうことになる (たとえば推測されやすいパスワードを使ってシステムにログインしているなどの場合が典型的である)。

このため、安全性の問題は「ユーザに適切な行動を考えて取ることを求める」という点で情報倫理の問題と類似しており、しばしば一緒に取り上げられる。

また、ユーザの行動が不適切でシステムの安全性が損なわれると、多くの場合は他の人にも迷惑が及ぶため、「他人の嫌がることはしない」という情報倫理の原則に照らしても、そのようなことは避ける必要がある。

しかし、このような類似制はあるものの、情報倫理と安全性は基本的には別の問題であり、きちんと区別されるべきである。具体的には、「あることをしてはいけない」という決まりがあった場合、その根拠が「正しい/正しくない」に由来するようであればそれは情報倫理の問題であり、「安全性を損なう」が理由であればそれは安全性の問題だといえる。

演習 「コンピュータやネットワークでやってはいけないこと」を 5 つ考えて紙に書き出し、それぞれの理由とそれが情報倫理の問題なのか安全性の問題なのか、またはさらに別の問題なのかを考えてみよ。

安全性に関わる問題の分類

安全性に関わる問題には、自分の力では防ぐことが難しいものもあるが、多くの問題は自分で注意することで防ぐことができるか、または被害を軽減することができる。ここでは、どのようなことがらが安全性の問題につながるかを整理してみる。

ユーザの行動 システムそのものには何ら問題がなくても、ユーザの行動によって安全性が損なわれることは多くある。その代表例は、上であげた「推測されやすいパスワード」であるが、それ以外にもパスワードを紙に書いて放置したり、ログインしたままの機器を放置したりして不正使用につながることがある。また、自分の個人情報を不用意に Web などで公開してしまったり、好奇心から出会い系サイトに行って知り合った人と会ってしまうなどの場合は、システムではなくその本人の安全がおびやかされることになる。

ユーザのエラー ユーザが「何が正しいか」分かっているのについ正しくない行動を取ってしまったり、正しい行動を判断できる十分な情報や考える時間を与えられない状態で選択を迫られたために正しくない選択をしてしまい、安全が損なわれることもある。このような場合は「ユーザミス」などとも呼ばれ、間違いをおかした人間が責められることが多いが、本来はユーザに責任がない場合も多い。人間はどんなに注意しても必ず過ちをおかす生き物であるし、自分の能力を超えた判断は正しく行えなくて当然なのだから、そのような状況にユーザを置いたこと自体に問題があると言える。すなわち、人間に判断を求める場合は的確な判断が行えるように分かりやすく情報を提示し、また間違いがあった場合でもそれを取り消してやり直すことができるようにシステムを作っておくべきである。もちろん、ユーザが本当にぼんやりしていて (あるいは飲酒などの結果) エラーをおかすこともないわけではないが、普通の状況でエラーの結果問題が起きたのであれば、その状況を改善しない限り同じ問題が何回も繰り返されることを忘れてはならない。

情報の信憑性 我々の行動は我々が受け取る情報によって影響されるが、それらの情報が常に正確であるとは限らない。このため、我々は常に受け取る情報の信憑性を意識し、重要な判断を下す根拠となる情報についてはその確かさを十分確認するべきである。たとえば架空請求詐欺やフィッシング詐欺は「まったくの嘘」であるが、それでも多くの人がだまされてしまう。電子メールや Web サイトの内容はそれを作る人がどのようにでも構成できることをよく覚えておくべきである。また、悪意を持った人が高度な技術を備えていて、暗号化や電子署名などの防御手段を破って (悪用して) 嘘の情報をつかませようとすることもある。

外部からの攻撃 外部からのシステムへの不正侵入の多くは、パスワードなどの不備によるものや、システムのセキュリティホール (欠陥、抜け穴) を利用したものである。これらを防ぐには、安全なパスワードを使用し、またシステムはきちんと更新して見つかった抜け穴がふさがれた状態を保つことが必要である。一部の攻撃は、ユーザをだまして攻撃のためのプログラムを実行させることに基づいている。ウィルスの感染などもこの種類に属する。これに対

する防御は、怪しいプログラムを決して実行させないように注意することに尽きる。

内部からの攻撃 システム犯罪のうち、多くの割合のものは内部者による犯行だというデータがある。たとえば、システム管理者が犯罪を企てた場合、それを止めることは非常に難しい。このような問題に対する対策としては、システム監査など防御面からのものと、組織を健全に保ち所属する人たちが意欲を持って働けるような環境を維持するなど予防面・組織面からのものとがある。

システムの不具合 今日のシステムにはソフトウェアがつきものであるが、規模が大きなソフトウェアを完全に正しく作ることは難しく、稼働後も何らかの欠陥が残っていることが普通である。また、ハードウェアは物理的に壊れることがある。これらの不具合によって、システムが必要な時に使えなかったり、重要なデータなどを喪失することは、安全上の大きな問題である。これに対処する方法としては、データは定期的に控え (バックアップ) を取って安全な場所に保管する、重要なシステムは予備を用意し障害があったときは切り替えて使えるようにするなどの方法があるが、コストも掛かり技術的にも簡単ではないこともある。

7.3　情報社会における出来事と考え方

ハイテク事件とその枠組み

ここまでにも繰り返し述べて来たように、コンピュータやネットワークという新しい媒体の発達とともに、これらに関係した、今までに経験したことのないような事件やできごとが我々のまわりで起こるようになってきた。これらの事件やできごとを総称して**ハイテク事件**と呼ぶこともある。

これらの事件やできごとを理解する枠組みについても、ここまででひととおり説明してきたが、簡単にまとめると次のようになる:

- 事件やできごとに登場する情報機器の機能や使われ方にどのような技術的 (および必要なら社会的) 特性があるかを考える。この部分がいわばハイテク事件の「ハイテク」部分の「味つけ」となる。

- 上記の「味つけ」を取り除いたできごとの本質として、それが「正しい/正しくない」の問題 (倫理の問題) なのか、安全性の問題なのか、両方なのか、どちらでもないのかを考える。
- 倫理の問題であれば、それが法律を破っているのか、マナー/モラルの問題なのか、スタイルや姿勢の問題なのか、実は問題ではないのかを考える。
- 安全性の問題であれば、ユーザの行動、ユーザのエラー、情報の信憑性、外部からの攻撃、内部からの攻撃、システムの不具合のどれに属する問題なのか、またそれ以外の問題なのかを考える。

安全性の問題であり、同時に倫理の問題でもあるものは多数ある。たとえば他者による攻撃や侵入は安全性の問題であるし、また攻撃や侵入を行う人は倫理的でないふるまいをしていると言える。

演習 以下に出て来る各事例について、それが情報機器のどのような技術的/社会的特性と関わっているか、また倫理的問題/安全性の問題としてどうなのかを分析せよ。

事例: 失われた労働

T 氏は M 社のソフトウェアを使って作業をしていた。3 時間掛かってようやく作業が終わってファイルに書こうとした時、突然ソフトウェアは異常終了してしまった。T 氏は途中の作業を何回かファイルに保存してあったので、再度ソフトウェアを起動してそのファイルを読み込もうとしたが、「壊れていて読み込めません」というメッセージが出て読み込めない。3 時間の作業が無駄になって怒りに燃えた T 氏は M 社に文句を言おうとしたが、ソフトの使用許諾契約には「ソフトの不具合によるお客様の損害に対して当社は責任を負わない」と明記してあった。ガーン。

この事例の技術的要因はコンピュータのソフトウェアである。ソフトウェアに欠陥がつきものであり、そのために作業の一部が無駄になることはあり得ることである。T 氏はそのことを知っていたので、こまめに途中の状態もファイルに保存していたのだが、ソフトがファイルに書き込んでいる途中で不具合が出た場合にファイルが部分的に書き換えられて矛盾した状態 (壊れた状態) になることにまで十分注意していなかった。そして、同じファイルを繰り返し上書きして保存していたので、最

後に不具合が出た時にその 1 つ前に保存した状態も合わせて壊してしまった。2 つのファイルを交替で書き換え保存し、保存するごとにソフトを立ち上げ直してファイルが読み込めることを確認しておけば、1 つ前のファイルは残っていたはずだが…(ソフトの欠陥の内容によっては、問題なく書きこんだはずのファイルが壊れていて読み込めないこともあるので、読んでみることも必要である)。

この事例では倫理的な問題はないと言える。ソフトウェアには欠陥がつきものなので、欠陥による不利益を補償しないという内容の使用許諾契約は一般に使われている。作業が失われるというのは安全性の問題である。その分類としてはシステムの不具合に関係があり、またそれを避けるためにはユーザの行動も改める必要がある (不具合を皆無にするというのは不可能であり現実的な解決策ではない)。

事例: 振り込め詐欺

ある日 60 歳を越える高齢者に電話がかかってきた。電話に出ると「俺だよ、俺」と言う。高齢者は孫が久しぶりに電話をかけてきた思い、「○○ (孫の名前) かい?」すると「うん、○○なんだけれども、今交通事故を起こして、示談金を支払うためにすぐに大金が必要なんだ。だから銀行口座へお金を振り込んで欲しい。」と説明した。また、「両親には怒られるから内緒にして欲しい。」と言う。高齢者は孫の為と思い指定された銀行口座へお金を振り込んだ。翌日孫にお金は間に合ったのかどうか確認するために電話をしてみると、「私交通事故なんて起こしてないよ?」と孫は言う。高齢者はお金をだまし取られてしまったのだ。

この事例では技術的要因は 2 つある。1 つは電話という情報機器であり、その「声だけで相手のようすが見えない」という性質がこのような詐欺を可能にしている。もう 1 つは銀行のオンラインシステムで、瞬時に送金ができて送金されたお金もすぐ引き出せるから犯人はお金を手に入れ逃走できる。以前であれば送金には数日掛かったので、犯人がお金を手に入れる前に気付いて送金を止められるのでこのような犯罪はほとんどなかった。

詐欺は犯罪であるので、この事件は倫理的問題がある。また、お金の損失は安全性の問題でもある。問題の種類としては、外部者による攻撃である。対策としては、銀行の送金金額の上限を小さくしたり (その分

だけ不便になる)、このような詐欺が増えていることを広報して注意を促すことなどが行われている。

事例 2: 東証 1 円 61 万株事件

2005 年、ある証券会社が人材派遣会社の株式を「61 万円で 1 株売却」とするつもりが、間違えて「1 円で 61 万株売却」として売り注文を出してしまった。誤注文に気がつき取り消しをしようとしたが、東証システムの不具合が原因で取り消しは実行されず、この異常に安い株を買った客が多数出た。証券会社は契約を履行するために実際にこの値段で客に株を渡すかそれができなければそれに見合う補償をする必要があり、300 億円以上と言われる損害を出した。

この事例の技術的要因は、株式等の売買がすべてコンピュータによって処理されるようになってきたこと、およびそのような高額の取り引きを処理するソフトウェアであっても欠陥は避けられないということがある。また、技術的でもあり社会的でもある要因として、市場側が「注文データは証券会社の担当というプロフェッショナルが投入するので、どのような極端なデータであってもそのまま処理する」という方針を採っていたことがある。

この事例には倫理的な問題はないが、証券会社が巨額の損失を出したという点で安全性の問題がある。その種類としては、間違った注文を出したというユーザエラーの問題がまず挙げられる。人間は必ず間違いをおかすので、このようなことが起きるのは避けられない。これに対して、証券会社内のシステムでは「異常な注文なのでそのまま出してもいいのか」という警告が出るようになっていたが、実際にはこの警告はしょっちゅう出るものなのでユーザは「OK」を選ぶのが習慣になっており、そのため異常な注文がそのまま出されてしまった。これは、システムがユーザに「ちょっと標準から外れた注文」「極めて異常な注文」という重要な違いをうまく伝えるようにできていなかったことに 1 つの問題があり、また、取り消しの処理ができなかったことにもう 1 つの問題がある。これらはいずれも (種別は違っているが) ソフトウェアの不具合と言ってよい。また、このような状況を改善しないままシステムを使い続けたユーザ側にも責任がある。

事例 3:チェーンメール

「家族が事故に合い、○×病院に運ばれました。しかし、出血が多く血が足りません。このメールを読んだ方で血を分けてくれる方を募集します。無理でもこのメールを 5 人の人に送ってもらえれば家族は助かると思います。家族のためにもこのメールをあなたで止めないで下さい。」といった様な内容のチェーンメールが流行った。メールをみた多くの人が人助けだと思い病院に駆け込んだり、病院に連絡をしたり、反響は大きかった。しかし、その病院には血液製剤が足りないような事態は起きていず、誰かのいたずらであった。そのため、病院ではその対応に追われ、一時病院の機能が麻痺してしまった。

この事例の技術的背景としては、電子メールが多くの人に使われるようになったこと、またメールのメッセージがごく短時間で相手に到達し、複数送信するのにさほど手間が掛からないことが挙げられる。このため、多くの人はつい「気軽に」チェーンメールの増殖に協力してしまい、極めて短時間で大量のメッセージが発生してしまう。

このメールを最初に出した人はチェーンメールのもたらす影響をよく知っていて出したものと思われる (「人助け」「緊急」など多くの人がつい協力してしまうような場面がわざと設定されている)。このため、この事例には倫理的問題があると言える (特定の病院を狙ったという点で威力業務妨害という法律違反に当たる可能性もある)。また、病院にとっては業務が妨げられたという点で安全性の問題があり、その種類としては外部からの攻撃といえる (多くの「善意の人」が結果的に外部からの攻撃に加担している)。対策としては、チェーンメールの問題について十分広報することであろう。

事例 4:フィッシング

金融機関等からお知らせメールがユーザーに送られてくる。そのメールの中にリンクが貼られており、普段見慣れた画面に、名前やユーザー ID、クレジット番号などを確認のために再入力するように指示されている。これは金融機関を装って送られた偽メールの例である。リンク先の Web ページは本物にそっくりの偽ページであり、うっかり入力してしまうと名前などの個人情報が盗まれてしまう。

この事例の技術的背景として、Web ページの内容はディジタル情報であり、本物そっくりの偽物を作ろうと思えば比較的用意に作ることができる、という点が挙げられる。また、このようなサイトではアドレス欄 (ページの URL が表示される) を隠して、本来の企業のドメイン名を持ったサイトでないことが分かりにくくすることが多い。

フィッシングは詐欺であり犯罪であるため、倫理的問題は当然ある。また、クレジット番号などを盗まれることは、安全性の問題でもある。その種類としては、外部者による攻撃でもあるが、「情報の信憑性を確認していない」というユーザの行動の問題でもある。また、このような情報を金融機関等がネット経由で問い合わせてくることはないという知識も持っていて然るべきだとも言える。

事例 5: 著作物の利用

ある遊園地について、旅行に行った体験記などを、個人の Web ページに写真や画像と共に掲載しようとした。写真を撮ったがあまり Web ページに乗せられるほど、満足のいく写真が無かった。そのため、google でイメージ検索で遊園地のキャラクターの画像を見つけてきて、個人の Web ページに掲載をした。数日後、Web ページに掲載してあった電子メールアドレスに 1 通のメールが届いた。そこには、あなたの Web ページは著作権を害しているため、すぐに画像の掲載をやめてください。」と警告されていた。

技術的問題としては、今日では Web 上に多くの画像があふれており、また検索技術の向上により見たい画像を探して来ることも容易になったという点が挙げられる。ただし、そのような画像の多くは他人の著作物であることも忘れてはならない。

「遊園地のキャラクター」や「他人 (この場合は遊園地) が撮影した写真」もまさに著作物であり、見つけて来た画像を個人的に楽しむのはかまわないが、Web ページに載せて公開することは著作権法違反になる。法律違反なので、倫理的な問題があるといえる (法律以前にも、他人の創作物を無断で利用することは「正しくない」と言えるだろう)。安全性については特に問題は見当たらない。

事例 6:P2P ファイル共有システム (著作権の侵害)

P2P システムを用いた、ファイル共有ソフトを利用していて、たまたま検索した結果に、通常ならば有償のソフトや音楽ファイル、ビデオ画像やアプリケーションがあった。これは便利だと感じ無料で利用していたところ、P2P システムを通じて有償のソフトや音楽ファイルを無断で公開した人が著作権法違反で逮捕されたというニュースを見て恐くなった。

この事例の技術的側面としては、P2P システムでは「草の根」的に通信が行われるので、著作権法に反する内容のものが流通していても「どこがそれを無断で公開している」と名指ししにくいことがある。逆に言えば、P2P システムを動かしている多くのユーザが著作権法に違反する行為に加担しているとも言える。社会的側面としては、このような「誰がやっているとも分からない」システムではユーザの自制が働きにくく、つい著作権法に違反することを許してしまうという面がある。

もちろん、著作権法に違反する行為は倫理的に問題があることは間違いない。また、自分のコンピュータが自分の意図しないところで著作権法違反に加担しているとすれば、それは安全性の問題でもある。その種別としては外部者の攻撃とも言えるが、半ばそのような行為が行われると知っていて P2P ソフトを動かすのであればユーザの行動の問題と考える方が適切であろう。

事例 7: 耐震強度偽装

マンションを建築するためには、設計段階で耐震強度を計算し、資料として提出しなければならない。しかしながら、ある建築家は、耐震強度の項目を計算した後に、その数値を自分に都合がいいように改竄した資料を提出した。さらに、その資料を国が検査を行ったが、きちんとした審査を行なわなかったことから検査に合格し、マンションが建てられてしまった。

この事例の技術的背景としては、ディジタル情報は電子署名など何らかの手段を取らない限り、自由に修正が可能だという問題がある。このため、正規の計算プログラムが出した結果をそのまま印刷する代わりに改ざんしてから印刷しても、見た目は正規の結果であるかのように見え

てしまったわけである。このような問題を避けるためには、改ざんが困難になるような仕組みを採り入れることも考えられるが、むしろ計算の元になるデータを提出させ、受理側でそのデータから計算を行って確認する方が筋が通っていると言える。

この事例は不正に建築確認を取得したという点で犯罪であり、倫理的問題がある。また、正しくないデータのまま強度が極めて小さいマンションが多数建てられ、多くの人が危険にさらされたり多額の金銭的損失をこうむったという点では、安全性の問題もあると言える。その種類としては、情報の信憑性であり、また本来なら建築物を適切に設計/チェックする立場の者が行ったという点で内部からの攻撃とも言える。

事例 8:(コミュニケーション能力の低下)

近年の情報通信ネットワークを利用したメディアの出現により、顔の見えない (相手を知らない) まま出会い、コミュニケーションを行う機会が非常に多くなった。相手が目の前に居ない、目の前にあるのは携帯電話やコンピュータのみ。という場面が増えることで、メディアの向こうにいる相手の事を気遣うことを知らない人が増えてきている。情報通信ネットワークの利用が盛んになる事で情報過多やコミュニケーション手段の多さ豊富さが、逆に対面対話能力を下げているのである。つまり今の私たちは、多くのコミュニケーション手段に振り回され、「コミュニケーション漬け」の毎日を暮らしていることになる。

この事例の技術的背景としては、情報通信ネットワークを用いたコミュニケーションが多様化し、また広く使われるようになっているという点が挙げられる。

相手のことを気遣うことができない人が増えるという問題は、倫理的な問題であるが、より正確に言えば「倫理的問題を引き起こすであろう要因の増大」という 1 段上の問題である。安全性の点も同様に、適切に情報を使いこなし人間的に生きることができないのは「安全性の問題を引き起こすであろう要因の増大」という 1 段上の問題である。そして、多くの倫理的問題、安全性の問題が引き起こされる下地が作られるという点で、直接の倫理的/安全性の問題よりも重大だと言える。

第8章　情報システムと情報社会

8.1　情報システムとその形態

8.1.1　情報システムの定義と由来

情報システムを単に「情報を扱うためのシステム」と考える立場もあるが、ここでは情報システムを次のように捉える:

定義 情報システムとは、情報の利用を望んでいるあらゆる人々にとって、手に入れやすく、役に立つような形で、組織体や社会における活動を支え、適切な情報を集め、保管し、加工し、伝達するしくみである。

すなわち情報システムとは、コンピュータを中心とした単なる機械的なしくみではなく、人間を含む社会的なシステムであって、社会活動に馴染むものである。

そのような情報システムはどのようにして生れ、育ってきたのであろうか。ここでいう情報は、コンピュータを使って手に入れるものとは限らない。大昔から人間は食べ物の在り処を探して情報を求め、危険を知らせる情報に耳を傾けてきた。つまり、生きるために必要な情報を入手してきたのである。

人間が集まると自ずと約束事ができ、グループの情報活動がはじまる。それは人間を構成要素とする「情報システム」である。われわの生活環境では、人間同士の情報伝達だけでなく、自然界との情報のやり取りも行われている。このときの情報伝達には、言葉の他に音や光や匂いなど、人間の五感を刺激する媒体が介在する。いずれも、コンピュータが出現する以前から実在していた。

最初のコンピュータは言葉通り「計算をする道具」として生み出されたが、その後、技術者たちは情報の蓄積と伝達の道具としてコンピュータが利用できることに気づいた。集めた情報を加工して蓄積し、必要な

ときにそれらを利用するという仕掛けを生み出したのである。情報システムとは、このように人間が創りだした人工物である。

ここでは、情報システムは人間活動を含む社会的なシステムであると捉え、この活動を支えるしくみの一つにコンピュータを位置づけている。情報システムはコンピュータの出現によって、人間に豊かな情報空間をもたらしたといえる。

8.1.2　情報システムのさまざまな形態

文書/資料管理システム

情報社会で日常的に利用されている情報システムに目を向け、それらがどのように形成され活用されているのかについて考えよう。

今日では、情報システムの殆どにコンピュータが含まれているがその形態は利用環境によって異なっている。

情報システムには、標準的なパッケージを選択して構成しているものと、特定の目的に合わせて独自に開発するものとがある。いずれの場合も、利用者の環境に適した規模と構成で実現される。したがって、目的が類似していても、環境や社会的な背景が異なれば情報システムの形態は違うものとなる。この立場の違いについて、文書や資料の管理システムを例として見てみよう。

公共機関である国立国会図書館や大学図書館では、学術雑誌等の出版物の電子出版化が進んでいる。さらなる電子図書館サービスとして、ネットワーク上の情報資源の蓄積と流通の仕掛けに注目している。これらのシステムは、従来のように個別の図書館ごとに物理的な「本」を管理するものではなく、多数の図書館の情報を連携させて扱い、それらを横断的に検索することを可能にする (図 8.1)。

企業では、組織活動で生まれた情報をディジタル化して蓄積管理し、ビジネスや個別業務の活動で利用する情報システムに注目している。他方、個人的なシステム環境では、調べごとをしたいときにネットワークを介して容易に情報を検索できるしくみを利用したいと考えるであろう。

このように、文書や資料を管理するシステムという点では同じであっても、それを必要とする場面や利用目的はさまざまであり、情報システムの姿もそれぞれ異なったものとなる。

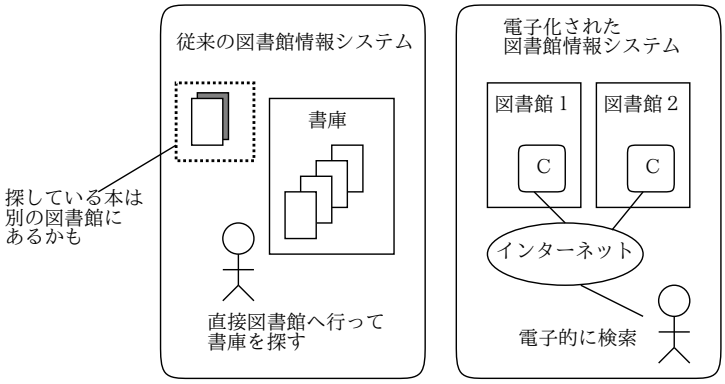


図 8.1: 図書館情報システム

災害と情報システム

自然のシステムと関係した情報システムも多い。過去の災害から得られた教訓を基に、生活に役立ついろいろな人工システムが開発されている。たとえば、阪神淡路大震災がきっかけとなって、広域災害における速やかな救援・救助のための情報提供が必要であると認識された。

その結果、自治体には、医療機関・消防機関・保健所等を結ぶ情報ネットワークシステムが導入された。組織に閉じていた病院情報システムが地域医療情報システムや広域災害救急医療情報システムと連携したり、その延長線上にある全国規模のバックアップセンターと連携したりと、いろいろな形で発展している (図 8.2)。

ライフラインと情報システム

日常的な生活環境で役立っている情報システムの例として、安全で良質な水を安定して供給することを目的とした、出水・送水・浄水・配水を管理する上水道総合管理システムがある。電力会社では送電・変電・配電の自動化システムを構築し、電力系統や給電設備を情報システムで監視するなど、電力の安定供給と危機管理を行なっている。

これらはライフラインを支える情報システムであり、平常時と非常時とを問わず重要な役割を担っている。そのため、トラブルが起こっても全体が致命的な運用停止などに陥ることがないように工夫されている。

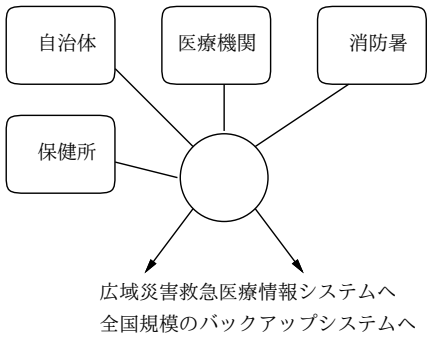


図 8.2: 防災システム

たとえば、設備管理システムのサーバを二重化するとか、広域ネットワークを利用して多重化するなどして、重要なデータを安全に共有できるしくみが考えられている。また、遠隔監視によって緊急時に離れたところからでも操作ができるようなシステムも稼働している。

都市ガスの安全対策のためのリアルタイム地震防災システム、大規模災害時の停電対策として開発された通話支援システム、被災地の復旧救援隊の専用連絡手段を確保する緊急衛星通信システム (通信衛星 N-STAR の活用) などもライフラインを支えるシステムである。

8.1.3　社会活動における情報システムの役割

社会活動や生活を支援する情報システムは、国や地方自治体といった官公庁が管轄するものが多い。たとえば、社会基盤的なシステム、公共活動を支えるシステム、小中高等学校の教育活動を支えるシステムなどがある。

図書館、博物館、美術館などの文化施設、あるいはスポーツ施設やレクリエーション施設などでは、利用者の利便性を配慮したサービスを重視している。これらのシステムでは、インターネット上の Web や E メールを介して施設の利用案内やイベント情報などを提供している。

自治体は住民登録・印鑑登録・外国人登録・国民健康保険・介護保険・住民基本台帳 (コラム 1 参照) などの管理システムを整備したり、生活保護の受給などの福祉支援システムを導入したりしている。そのため行

(コラム 1)
住民基本台帳ネットワーク (住基ネット) システムは 2002 年 8 月にサービスが開始された。これにより、全国のどこの市町村からでも住民票を取得することが可能になり、また、住居変更の手続きを一回するだけで自動的に各種の手続きができるようになった。ワンストップサービスという。このしくみは、基本 4 情報 (氏名、生年月日、性別、住所) と全国共通の「住民票コード」を「全国センター」で保持することによって実現できた。
また住民情報システムによって、住民票や戸籍抄本の取得、結婚届や出生届などが簡単になり、生活保護や障害福祉や児童手当に関係する住民データの更新や管理も容易になった。

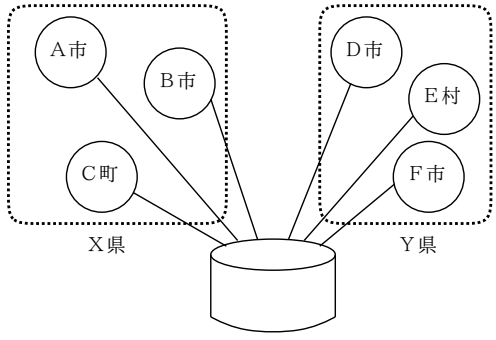


図 8.3: 住民基本台帳の利用

政を支援するたくさんの情報システムが稼働しているが、それらは業務の効率化と住民サービスに役立っている。道路地図や上下水道の図面・防災エリア地図などもディジタルデータで保持し、部門内に限らない横断的な活用ができるようになっている。

社会的な基盤に関係するものとしては、以上のほかに福祉施設・病院・保健所・消防署・警察署などの業務を支援するシステムがある。また、火山噴火・土砂災害・河川水量の監視などの防災システム (コラム 2 参照)、大気汚染・河川水質の監視などの環境システム、広域災害緊急医療情報システム、消防通信指令システム、交通管制システムなどもライフラインの確保に貢献している。

教育環境では、学校と都道府県や市町村と教育センターなどの間で、情報が共有されている。各学校では、施設・設備の管理、教職員や児童生徒に関する個人情報の管理、視聴覚室や情報処理室の利用に関する諸情報の提供、ネットワークや学校の Web サイトの整備など、教育活動を支援するさまざまな情報システムが稼働している。学校が独自に開発している各種教材のデータベースや教材検索システムもある。これらの教材資源の多くは、授業や課外活動で、教師によって、あるいは児童や生徒が調べ学習をするために有効に利用されている。

8.1.4　組織の情報システムと個人の情報システム環境

情報社会における組織は、いろいろな形で形成されている。たとえば、人が一箇所に集まって活動する形の組織もあれば、離れたところに分散しながらネットワークで繋がって活動している組織もある。組織を構成する人々の相互の関係に注目すると、いずれの組織も情報システムととらえることができる。これは広い意味での情報システムである。

情報システムとして見た場合、組織は、それぞれの組織の業務に関する情報を処理し、組織の中と組織を取り巻く環境での動向や異常を検知し、それに基いて行われる意思決定を支援し、その決定を伝達するという役割を担っている。組織における情報には、組織全体で共有されているものと、組織の中の一部門で共有されているものとがある。

共有される情報は、組織のルールにしたがって管理されているが、それらの情報を扱うのは組織の中の個人である。組織の情報システム環境にコンピュータが取り入れられるとき、組織のルールはコンピュータによる処理にも適用される。

(コラム 2)
災害対策支援では行政と情報サービス産業が協力して、いろいろな状況を想定した情報システムを開発し、緊急時に備えて運用している。たとえば火山噴火への対応では、火山性微動や火山性ガスなどを常時計測し、噴火の恐れがある場合に避難警報を発している。
洪水の恐れがある地域では、情報システムを活用して河川の水位を遠隔監視し、ダムの放水や下流の水門制御などを行っている。土砂災害が発生しやすい地域では、住民の生命を守るためにシステムが稼動しており、住民に対して非難警報などの適切な情報を速やかに出せるようになっている。
地震などの広域災害が発生した場合は、通信手段にも影響が現われる。固定電話・携帯電話・インターネットが使えなくなる可能性が高いため、災害の情報などを早く正確に伝えるために、これらとは別の防災通信ネットワークシステムが構築されている。
このような防災システムに優れた監視システムが備わっているにもかかわらず、行政の管轄の違いにより住民に適切な情報が伝わらないというトラブルが発生したことがある。このため、これらのシステムの構築にあたっては、国と地方自治体との連携方法や責任と権限をどう持たせるかということについて、考慮されるようになった。

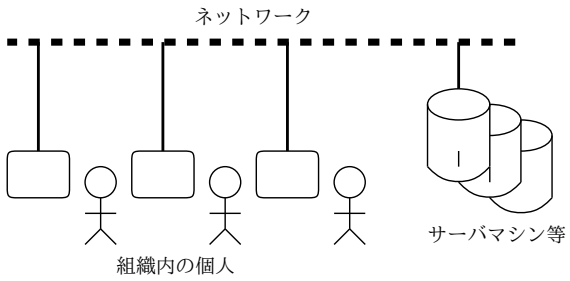


図 8.4: 組織の情報システムと個人の利用環境

組織で共有される情報が整理され、コンピュータに蓄積されていれば、組織のメンバーはその情報環境にアクセスして必要な情報を入手することができる。この活動を効率よく行うために、個人の情報システム環境を整備しておくことが必要になる。

コンピュータやその周辺機器が安価になった今日では、組織において一人一台のコンピュータ (パソコンなど) が配備されていることが多い。同じ組織に配属されていてもそれぞれに与えられている役割は異なるので、組織の中の個々人に許可された情報システムを効果的に活用できるように、パソコン環境を整備することが必要になる。これらの環境を整備し、維持・管理するのは、最終的には利用者個人である。

アクセス権限を効率的に利用しつつ、一方で組織全体の安全性を維持しながら業務を適切に遂行するために、利便性を配慮した汎用のアプリケーションを導入することも必要になる。その選択と責任は利用者それぞれにあり、その結果が個人の情報システム環境として形成されることになる。

8.2　企業における情報システム

8.2.1　企業活動と情報システム

企業の情報システムが持つ要素

企業の情報システムは、最終的には企業が利潤を追求するために開発されるものであるが、その目的は、業務の効率を上げること、ビジネスの改革をすること、他者と情報を共有すること、他社との競合を有利にすることなどいろいろである。情報システムの構築の方式は組織ごとに違っているが、基本的には、

- 組織に共通なプラットホーム
- 業務に特化したアプリケーション
- 組織で共用するデータ資源

の 3 種類の要素で構成されている。

販売情報と POS システム

コンビニエンスストアやスーパーマーケットでは、商品を販売するとき販売時点の情報を管理する POS(point-of-sale) と呼ばれるシステムを利用している。このとき、商品情報が記号化されたバーコードを商品に貼付することによって商品管理を容易にしている (コラム 3 参照)。

このような POS システムを導入したことで、レジで商品の金額を入力する手間が簡素化され、入力ミスも激減した。また、商品価格の変更が簡単になり、価格の割引などをする際の精算の自動処理も容易になった。

コンビニエンスストアではレジでの精算時に、商品のバーコードを読み取らせるとともに、店員がおおよその年齢と性別を区別するボタンを押している。これによって、店舗ごとに「どの売上時間帯にどんな層の客がどの商品を購入したか」を情報として収集でき、店舗ごとの商品の陳列戦略を立てることが容易になった。

POS システムを活用することで、各店舗の立地環境や売れ筋商品、時間帯による売上傾向、消費者層、地域のイベントや気温・天候、などと関連した売れ筋商品などの情報を詳細に採取することが可能になり、販売戦略の意思決定に役立っている。さらに、採取した情報を共有し配送センターや製造会社とも連携することによって、必要な商品をきめ細かに補充することも可能になった。

(コラム 3)
POS システムは 1982 年にセブンイレブンで初めて構築され、以後システムの改善が繰り返されてきた。今日では、商品取引における販売情報を活用した総合店舗情報システムとして、精算や売上管理、在庫管理も自動的に行えるようになっている。
セブンイレブンでは 1984 年に POS レジスタで扱う商品にバーコード (異なる太さのバーを組み合わせて数値を表現するもの) を付けるようになり、その後、幅広く百貨店やスーパーマーケットやコンビニエンスストアに普及するきっかけとなった。バーコードで表示されている商品番号と、商品名や価格の対応一覧をデータとして蓄積しておくことによって、該当する商品の名称や価格をレシートに印刷できるしくみになっている。

組織活動を支援する情報システム

組織活動を支援する情報システムという切り口では、企業、行政、あるいは公共の組織が対象となる。企業に注目すると、品質の高い製品を安価に提供することによって収益を拡大するために、組織内の業務を効率よくこなすシステムが考えられている。

企業ごとに相違はあるが、製品を製造する製造部門、製品を販売する営業部門、お金の管理を行う経理部門、業務に必要な備品や消耗品を扱う購買部門、従業員の採用や配置に関わる人事部門など、各々を支援する情報システムが稼働している。

組織では、別の部署で同じ情報を利用することがあるため、データの入力や蓄積が二重にならないように配慮している。また、企業全体として効率よく業務を行えるように、ハードウェアやソフトウェアの配置も工夫して全体システムを設計している。

一方、業務のやりかた (プロセス) に注目すると、どの組織にも同じような業務処理のしくみがある。そこで、一般的な業務プロセスをパッケージ化した統合業務システム (ERP) が開発されている。この ERP を導入する企業が増えているが、効果的に活用するためには、予め自組織の業務プロセスを分析し (時には見直すことも必要になる)、業務環境に適したパラメータを設定しておくことが重要になっている。

8.2.2　商取引と情報システム

B2B、B2C、C2C

インターネットの普及によって, ネットビジネスと呼ばれる、インターネットを利用した新たな商取引の形態が出現した。銀行のオンラインネットワークは、当該銀行に固定設置された ATM 端末でサービスを受ける形態から、家庭のパソコンやコンビニの ATM 端末からサービスを受けられる自由な接続形態へと発展した。電車や旅客機の座席予約や宿の予約を、インターネットを介して手配することも可能となった。

インターネットを活用した商品販売、株式売買、オークションなどの新しいビジネスモデル (商売の仕組み) が続々と生まれている。企業間の取引 (Business to Business:B2B)、企業と消費者間の取引 (Business to Consumer:B2C)、消費者同士の取引 (Consumer to Consumer:C2C)

などと呼ばれる電子商取引がある。これらの取引には、インターネットサービスプロバイダ (ISP) が介在する。

これらのうち、われわれになじみが深いのは B2C、すなわち企業がネット上でさまざまな商品を消費者に販売する形態だと言える。インターネット上で商品を売買する Web サイト (ネットショップなどと呼ばれている) には、商品の検索・注文・支払いなどを一括処理する情報システムが機能している。仕掛けはショップによって異なるが、基本的には、顧客が Web ページから商品を選択し、決済方法を指定して購入申込みをするという流れである。ネットショップで扱われている商品は、書籍, 家電製品、日用品、食品, 保険、冠婚葬祭など多岐にわたっている。

Web サイトに電子商店を集めたオンラインモール (電子商店街) では、複数店舗の商品を比較したり、複数店舗の決済や配送などを一括処理したりといった複合的なサービスを受けることができる。

宿泊予約サイトの情報システムは、「宿泊施設が宿泊予約サイトに空き室を提供する。利用者は宿泊予約サイトで宿泊日、宿泊地域、価格帯、サービスなどの希望条件にあった宿泊施設を検索し、ホテルを予約する。宿泊予約サイトはホテルに利用者を紹介して、その仲介手数料をホテルから受け取る。」という仕掛けである。これによって、ホテルは集客が多くなり仲介手数料を払ってもなお増収になる、ということで低料金の設定が可能になったという。

一般に、情報システムが介入する電子商取引には、「人が直接介在しない、顧客の参加・退出が簡便である、双方向性がある、グローバルな取引である」などの特徴がある。

なお、Web サイトを用いた商取り引きには安全性のための配慮が不可欠である。具体的には、次のことが求められる:

- 取り引きを行おうとするサイトが確かにその企業なり取り引き相手のサイトであることを確認する。
- 取り引きのためのやりとりが第 3 者によって盗み見られたりしない。

これらのことを実現するために、**SSL**(Secure Socket Layer) と呼ばれる技術が多く使われている (コラム 4 参照)。

(コラム 4) SSL と PKI

SSL は、(1) サーバが確かに信頼できる相手先かどうかを確認する機能と、(2) 通信内容を暗号化して盗聴を防ぐ機能とを提供している。前者のために、企業は**認証局** (CA、Ceritfication Authority) と呼ばれる組織に自分の所在等を証明する書類を提出し、CA はディジタル署名したその企業用の証明を作成し送付する。企業が Web サーバにその証明書を設置することで、ブラウザで SSL 接続したときに証明書がブラウザ側に提示される。ブラウザは主要な CA の照合用証明書を最初から内蔵しており、これを使って提示された証明書が本ものかどうかをチェックする。照合が成功しなかった場合はブラウザ画面に「提示された証明書が正しいことが確認できない」と警告が出るので、その場合はサイトの利用を中止するべきである。この、CA による証明書発行のシステム全体は **PKI** (Public Key Infrastructure) と呼ばれ、SSL 以外にもさまざまな認証に使うことができる。

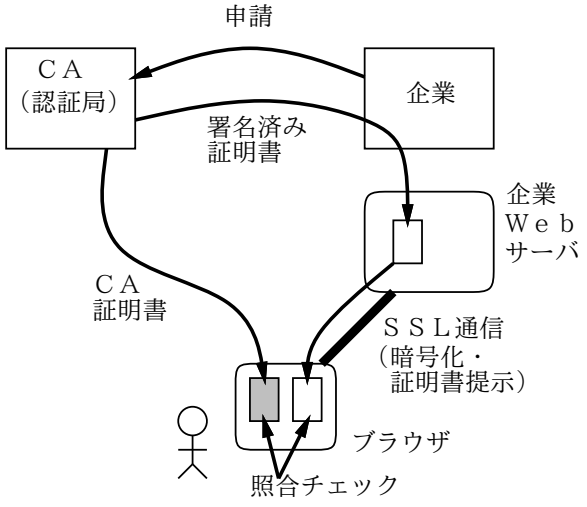


図 8.5: 住民基本台帳の利用

IC カードと IC タグ

ネットワークを活用したもう一つのビジネスモデルは、電子式カードシステムである。たとえば、利用情報を電子的に ID カードに記録しておき、管理システムが読取って速やかに外部のサーバと交信するという方式、あるいは記録した情報を管理システムが後刻読取って処理するという方式などが採用されている。ID カードには、磁気カード方式 (磁気ストライプでデータを記録する方式) と IC カード方式 (半導体チップをカードに埋め込んで利用する方式) とがある。それらは、利用者認証、プリペイド、ポストペイドなどのように、利用目的によって使い分けられている。IC カード方式には、リチャージ (積み増し入金) 機能が付加され、財布代わりに利用できるものもあり、これらは電子マネーと呼ばれている。

商取引に関する新しい情報サービスとして、バーコードや IC タグ (IC を内蔵したタグ) を利用した商品の追跡管理がある。トレーサビリティという。食品の偽装表示問題が発端となって安全性への関心が高まり、店頭で販売されている商品がいつどこで誰によって生産されたものか、どのような経路で店頭に到達したかなどについて、消費者の知る権利が話題になり考え出されたしくみである。

たとえば魚のトレーサビリティでは、個々の魚に IC タグが付けられ、獲れた場所、時刻、水揚げされた漁港、加工業者、流通経路などの情報が次々と記録され、消費者はその情報を知ることで安心して購入できる。また、宅配便サービスでは、荷物の伝票に記載された番号によって、荷物が何時に配達されたか、どこの集荷場を通過したかなどを宅配便業者の Web ページで追跡することができる。

8.2.3 企業等の人材育成における情報システムの位置づけ

企業教育と e ラーニング

インターネットなどを利用して躍進した教育システムとして e ラーニングがある。これはラジオやテレビなどの教養番組、衛星を利用した遠隔教育システムの延長線上のものである。e ラーニングの主流はインターネットを利用した教材の配信であり、時間や場所は受講者の自由に委ねられているという特徴がある。教材には、アニメーションやコンピュー

タグラフィックス、あるいは静止画や動画像を多彩に取り入れたものが多い。

学校で使われるもの以外の e ラーニングシステムは、予備校による衛星放送を使った遠隔教育に端を発し、その後企業の社内研修などに適したスタイルとなって普及した。

社会のルールが変わったり新しい技術が生れたりすると、企業は急いで従業員教育をする必要性が発生する。このような場合に e ラーニングを活用すると、遠隔地の従業員にも一斉に教育を実施することが可能になる。また、少ない講師でも短期間で教育を実施できるというメリットがある。

Web ページに学習教材をおき Web ブラウザを使って教材を見ながら学習する方式を、WBT(Web Based Training) と呼ぶが、このスタイルは利用者にとって便利で、自分のレベルに合った学習の進捗ができるというメリットがある。分からない時にその場で質問できないというデメリットもあるが、メールを使って質問や回答をする仕掛けを導入するなどの工夫もなされている。e ラーニング提供者は一度教材を作成すると、教材の内容が大きく変化しない限り長く使い続けられるので、投資効果が高いという。

人材管理とスキルの管理

人材のスキルを統合的に管理する情報システムもある。たとえば、個人の特性・スキル・知識などを定量的に測定し、資格・研修受講履歴・業務経歴などと一緒に統合的に管理する人材マネジメントシステムがある。経営者はこのシステムを使って、企業にどのようなスキルや知識をもった社員が何人いるのか、業務に必要なスキルや知識をもつ人材 (IT 人材: コラム 5 参照) をどのように採用するのか、どのように適材適所の配置をするのかなどに、このシステムを活用することができる。社員はまた、自分のスキルや知識を一覧で確認し、自分の強みや弱みを知ることができ、自己研鑽や目標設定の参考になる。

(コラム 5)
IT が、ビジネス、政治、個人生活などのあらゆる環境に浸透したことで、優れた IT 人材を大量に育成することが必要になった。一方では、国際的な競争が激化し、わが国の情報サービス産業の強化が求められるようになった。
このように環境変化に伴って、人材ニーズにも変化が現れた。これに対応するために、経済産業省は 2002 年に IT スキル標準 (ITSS) という IT 専門家の育成に有用な共通の枠組みをまとめて公表している。そこでは、情報サービス産業にどのような職種があるか、それぞれの職種に必要な人材のスキルとは何かなどが明らかにされている。2006 年には IT スキル標準 V2 も発表されている。

8.3 情報システムの構造と特性

8.3.1 情報システムの種別とそれに対する要請

事務処理と制御処理

利用形態に注目すると、企業における情報システムには事務処理と制御処理の 2 種類があることがわかる。

事務処理系のシステムは、人間が操作して情報を管理したり引き出す形の情報システムであり、処理の時間的制約は厳しくないが、人間が必要とする多様な処理に柔軟に対応できることが必要となる。そのようなシステムの例として、会計処理、営業処理、人事管理などがある。

制御処理系のシステムは、操作対象から直接センサーなどを通じて情報を読み取り、制御機器を通じて直接 (人間の介在なしに) 操作対象に働きかけるような情報システムであり、処理の内容はあらかじめ決められていて、短い時間内に応答することが求められる。そのようなシステムの例としては、化学反応の制御、センサーによる自動制御などがある。ほかに、炊飯器、調理器、洗濯機、自動車、エアコン、カーナビ、医用機器、デジタルカメラなど各種の機器に組み込まれた情報システム (組込システム) もこの種別に含まれる。

人間とコンピュータの協調

人間は組織や社会のしくみの中で情報を創造し利用し発信している。また情報システムは情報の創造と利用に密接に関わっており、人間系と機械系 (コンピュータなど) の機構で構成されている。つまり、情報システムは機械だけで構成されるのではなく、人間が不可欠な要素になっている。

情報システムは人間の社会があれば存在するが、コンピュータだけの情報システムはありえない。何故ならば、人間の関わりがなく、コンピュータが自ら意味のある情報を創造することは不可能だからである。すなわち、情報システムの要素である人間系と機械系は相互に密接に関係しあって情報を収集し、処理し、伝達し、利用するために協調して機能することが必要である。

情報システムは、さまざまな組織において、それぞれの目的で構築され運用されている。たとえば企業の情報システムでは、経営戦略との整合性が重視される。よい情報システムによって生産性や顧客サービスが向上し、それが顧客増加につながり、結果として業績向上に貢献できる。この仕掛けでは人間の行為や行動が優先し、それを支えるのがコンピュータである。そこでは人間とコンピュータの協調が重視される。

これらに加え、情報システムの構築および運用にも、人間とコンピュータが共に深く関わっている。

8.3.2　情報システムのしくみと特性

情報システムのさまざまな処理形態

情報システムは役割によって、処理形態が異なる。たとえば、入出金や座席予約や宿泊予約のシステムのように即時に結果を必要とする場合にはリアルタイム処理 (入力に対して短時間で処理を行い結果を返す方式) が適している。給与やガス・電気・水道・電話料金などの精算のように月極めで処理する場合にはバッチ処理 (一定量の計算をまとめて計画的に実施する方式) が適している。

設置場所に注目すると、システムを一箇所に集中して管理する方法と複数箇所に分散して管理する方法とがある。前者を集中処理方式といい、後者を分散処理方式という。コンピュータが高価であった時代は集中処

理方式がほとんどであったが、コンピュータや周辺機器の価格が下がるにつれて分散処理方式が増えている。

分散処理には、クライアント・サーバシステムといって、クライアント(サービス依頼者) とサーバ (サービス提供者) に役割分担して処理する考え方がある。一般に、複数のクライアントに対して一つのサーバがサービスを提供している。たとえば、データベースサーバやメールサーバなどがある。クライアントやサーバというのはもともと「役割」を表すもので具体的な機器を指すものではないが、それぞれ別々のコンピュータに担わせて「クライアントマシン」「サーバマシン」として使うことがよく行なわれている。コンピュータをクライアントマシンとサーバマシンに分ける方法を二層アーキテクチャ(構造のこと) と呼んでいる。

三層アーキテクチャ

情報システムの開発では、三層アーキテクチャがよく使われる。三層アーキテクチャには、3 つの異なる概念がある。それらは、ハードウェア配置の三層アーキテクチャ、システムソフトウェアの三層アーキテクチャ、そしてアプリケーションの三層アーキテクチャである。

ハードウェア配置の三層アーキテクチャとは、データベースサーバ用・アプリケーションサーバ用・ユーザインタフェース用にそれぞれ別のコンピュータを割当てることを意味する。これに対してシステムソフトウェアの三層アーキテクチャでは、これらのサーバの階層を意味し、別々のコンピュータを配置するとは限らない。

アプリケーションの三層アーキテクチャは、データ管理層・ビジネスロジック層・ユーザインタフェース層という 3 つの層からなる (図 8.6)。この場合も別々のコンピュータを配置するとは限らない。データ管理層は、データベースへのデータの更新・検索・追加・削除などの要求を処理する層である。ビジネスロジック層は、ビジネスに関するルールやビジネスのプロセスを処理する層である。ユーザインタフェース層は、クライアントとサーバに適用されるグラフィカルユーザインタフェース (GUI) を使って、データの入力やデータ表示を処理する層である。

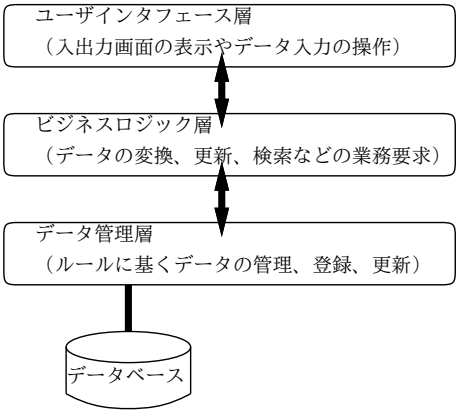


図 8.6: アプリケーションの 3 層アーキテクチャ

情報システムの計画と開発

システムモデリングでは、アプリケーションの構造とデータ管理を如何に適切にコンピュータ上に配置するかを検討することが重要になっている。

情報システムの開発では、開発費用と開発期間とソフトウェアの品質が重要な要件である。特に品質に関しては、ユーザが必要としている機能があるか (機能性)、正常に機能するか (信頼性)、使い勝手はよいか (使用性)、効率的に処理できるか (効率性)、機能の拡張や修正が容易か (保守性)、他のコンピュータへの移植は容易か (移植性) などが評価の対象となる。

情報システムは、開発の計画がなされ、開発され、誕生して運用され、廃棄されるが、それまでには、環境の変化を反映して繰り返し改善される。情報システムが生れてから破棄されるまでの生涯をライフサイクルと呼んでいる。

8.3.3 情報システムはどのようにつくられるか

情報システムのライフサイクル

情報システムは、利用対象が明らかであり、利用者ごとに固有の目的や仕様があって、それを反映して開発される。システム開発 (または再構築) のライフサイクルは、図 8.7 のような活動で示すことができる。

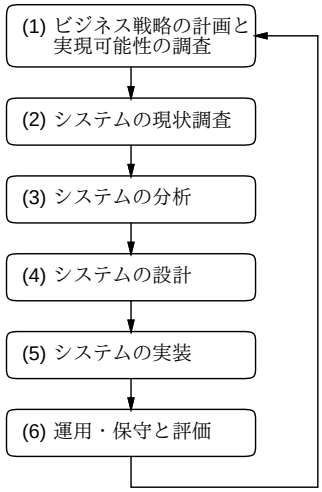


図 8.7: 情報システムのライフサイクル

図 8.7 は終わりのない繰り返しの絵になっているが、それは、組織が存続している限り情報システムそのものに終わりがないことを意味している。一般に、システムが改善されると、それまで使われていた機械的機構の不要になった部分のみが廃棄される。したがって情報システムが死ぬということは、全く新しいシステムに置き換わることを意味してきる。図 8.7 の項目について、もう少し説明を加える必要があろう (コラム 6 参照)。

情報システムの変更

情報システムにトラブルが発生しなくてもシステムの修正や変更は発生する。情報システムはビジネスの仕組みの中に組み込まれており、ビ

(コラム 6)
情報システムのライフサイクルの説明
(1) では、対象とする組織 (企業、部門などの利用者の集まり) が、情報システム運用上の課題を明らかにし、解決の可能性を検討する。このとき、予算・納期・システムの品質などの条件を明らかにする。
(2) では、現行のシステム環境において、問題状況と要求内容を調査する。その際、利用者が何を要求しているか、何故それを要求するのか、どのような資源を活用できるのか、どのような資源を必要としているのかを理解する。
(3) では、調査で得られた状況を分析して、要求を満たす代替案を二つ三つ検討し、制約条件を考えて、何を実現し何を犠牲にするのかの意思決定をする (トレードオフという)。
(4) では、導入する機能や構築するデータモデルを明かにし、新システムの仕様を決める。この段階で、これから構築する情報システム中のコンピュータに入力する情報・コンピュータから出力する情報・システムに蓄積するデータの内容や構造 (データベースやデータファイルなど) を決定する。この作業をモデリングという。
(5) では、ソフトウェアの仕様を作成し、実装するプログラムを作成し、コンピュータが仕様どおりに動くかをテストする。この活動をプログラミングという。
(6) では、新しいシステムを使って仕事を行い、トラブルの監視、トラブル発生時の対応などを行う。運用中に発生したトラブルの原因を解明し、プログラムをすぐ修復するか、次のシステム改善で対応するかを判断するのもこの活動の範囲である。

ジネスが世の中の変化に追随して変わっていくからである。たとえば、銀行の合併や買収、新しい商品やサービスの開発、法律の改正などに伴って情報システムも変更される。したがって、システムの運用や保守の段階では、絶えず稼動しているシステムの品質の評価を行い、次の計画につなげていくことが重要になる。

情報システムは、機能やサブシステムが追加・修正されて大きくなると、個々のサブシステムが最適に開発されていても、組織全体の構造が複雑になり全体として整合性のとれた最適なシステムとして機能しなくなる。情報システムの開発にあたっては、このことを念頭にいれておくことが必要になる。

新システムが順調に稼働する (実運用) と、それまで運用していたシステムは任務を終えてその生涯を閉じる。現実には、開発された情報システムが 10 年以上にわたって使い続けられることが多い。このため情報システムは、計画どおりの期間と費用で完成し、計画どおりの機能と性能を満たし、長期間にわたってビジネスの成果をあげ続けられるように開発することが重要となる。

このような情報システムを開発するための手法は世の中にたくさん発表されているが、どんなシステム開発にでも効果的に適用できるという方法はない。開発者は対応するシステムの特徴を理解して、手法を使い分けることになる。

開発と運用における関係者の役割

情報システム開発のライフサイクルで見てきたように、開発に当たっては、利用者と開発者 (関係者) が互いの役割をよく認識して、協力し合うことが大切である。

情報システムの開発では、情報技術に詳しいシステム開発の専門家と、情報システムを活用して業務を効率よく運営していく利用者 (経営者、業務担当者など) とが協力することが必要とされる。開発の前半では、利用者の視点でシステムのあるべき姿を明確にし、後半では、開発者の視点で効果的に実現する方法を考える。

情報システム開発の専門分野については、4.2.3 で述べた IT スキル標準 (ITSS) に示されている。業務に関する利用者の要望を調査しコンピュータ上で実現する仕掛けを分析し立案する IT アーキテクト、ネットワークやデータベースやソフトウェアを開発する IT スペシャリスト、

ソフトウェアの設計や開発を立案するアプリケーションスペシャリストなどと呼ばれる職種がある。

開発された情報システムが全体としてうまく働くことを確認することをシステムテストという。システムテストには、アプリケーションスペシャリスト (要求仕様書通りに作られているかをテストする) と、利用者 (システムが仕事を行うのに使い勝手が良いかどうかをテストする) とが関係する。実装が完了した後のシステム運用には、システム管理者 (コンピュータやネットワークやデータベースを管理・運用する技術者) とオペレータ (コンピュータの操作を行う人) が関係する。運用中のシステムの保守や改善には、利用者と協力してアプリケーションスペシャリストが対応する。

システムの利用者は、その情報システムの対象となる仕事の内容と必要な情報を調査・分析して、新しい仕事のやり方を立案する。その際、現在行っている仕事のやり方を見直し、効率よく課題を解決すために人とコンピュータがどのように役割を分担し、また協調するのかを考えることになる。

現状組織での分担にとらわれずに、業務のあるべき姿を描き、何をやるべきか (what)、いかにしてやるべきか (how)、誰がやるべきか (who)、いつやるべきか (when)、どこでやるべきか (where) を検討するとともに、なぜ導入するのか (why) を考えることが必要である。これらの新しい仕事のモデルを考えるのは利用者であり、仕事のプロセスを開発者に伝えることになる。この段階で業務のモデル化がうまく出来ていないと、システムが思い通りに機能せず、時には使えないものになってしまう。

開発プロジェクトの管理

一般に、情報システム開発のプロジェクトには大勢の関係者がかかわる。中には、何百人もの人が、数年かけて開発する大規模なものもある。このため、プロジェクトを管理する人が必要である。このような人はプロジェクトマネージャーと呼ばれ、全体の管理に責任を持つ。管理対象は、システムの品質と工期と費用である。プロジェクトマネージャーの主な仕事は、開発費用の見積もりと開発スケジュールを作成すること、プロジェクトのメンバーの確保と組織の編成を行うこと、計画どおりにシステム開発を完成するための管理を行うこと、開発関係者とのコミュニケーションを図ることである。

管理対象の一つである品質に関して、システムの機能と性能に関する信頼性の視点がある。それらは、仕事を行うために必要十分な機能を備えているか、仕事を効率よく行うための性能は十分か、システムがトラブルなく継続的に利用できるかのそれぞれの視点を表わしている。品質の保証は、システム開発時間とのトレードオフになることがあり、開発費用とのトレードオフになることもある。

情報システムの運用中に何らかのトラブルが発生することがある。これらのトラブルを予め想定して可能ならば回避をしたり、あるいは発生したときに速やかに対処したりすることが必要になる。社会的に影響が大きいシステムのトラブルを極力防ぐように設計することも重要である。

システムトラブルへの対策として、次のことが考えられる。ハードウェア障害の対策としては、待機系のシステムを設置しておき、トラブルの発生時に速やかに切り替えて業務を続行することが考えられる。操作ミスやテストの不十分さに関する対策は、担当者の教育や非常時の訓練を十分に行うことである。また、処理能力を超えたデータ量の増大によるシステム停止のトラブルを配慮して、さまざまな点で十分に余裕のある情報システムを計画しておくことが必要である。

8.4 データの管理

8.4.1 情報システムを支える情報とデータ

情報システムでは、いろいろな種類の情報やデータを扱う。情報やデータは情報システムの運用過程で生成され、蓄積され、利用される。これらの情報やデータをどのような形で保管し、維持するのか、どのように検索するのかを決めるのは、情報システム設計における重要な事柄である。

私たちは、情報とかデータという言葉をよく耳にするが、これらの使い方はかなりあいまいである。どちらを使っても意味が通じる場合と、なんとなく違和感をおぼえる場合とがある。情報システムでは、意図を明確にすることがしばしば求められるので、この言葉の使い分けも明確にしておくことが必要である。

「データ」は客観的な事実を表すもので、単なる記号の列として表現される。情報はデータに意味解釈を持たせたものであり、送り手から受け手に意味付けして伝達される。さらに、情報を一般的な概念として

体系的に整理されたものを知識と呼んでいる。これらの関係は図 8.8 のように表すことができる。この章でも、この定義に基づいて使い分けている。

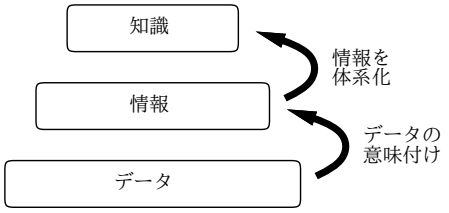


図 8.8: 知識・情報・データの関係

情報システムを支える情報には、情報システムに対して外部から入力する情報 (ここでは入力情報と呼ぶことにしよう) と情報システムから利用者に出力する情報 (ここでは出力情報と呼ぶことにしよう) とがある。入力情報には、人間が入力する情報と他のシステムまたは装置から取り込むデータとがある。

情報システムの内部で加工処理するデータも情報システムを支える重要なデータである。これらのデータの多くは、入力情報がコンピュータにわかる言葉に変換されたもので、処理過程で加工され、出力情報に変換される。システム内部で発生し、一時的に利用されて消えていくデータもある。

8.4.2　情報システムとデータベース

データベースとデータベース管理システム

組織の活動ではさまざまな情報が発生する。これらの情報はあるときは活動の成果として、またあるときは新たな活動の情報源として保管、蓄積される。その際、使いたいときに取り出しやすくするために一定のルールを定めて蓄積することが必要である。

このような、処理の手順や効率のためにルールを定めて蓄積された一群のデータの集まりを**データベース**、そのようなデータを効率よく安全

に管理するためのシステムを**データベース管理システム** (DBMS、Data Base Management System) と呼ぶ。[1]

DBMS が管理するデータには、次のような性質が見られる:

- 効率的に処理したい大量データを集め
- 組織にとって価値のある永続的なデータを保管し
- 複数のアプリケーションからアクセスされ
- 複数ユーザにより更新される

このため、DBMS は通常のファイルシステムとは異なる構造や機能を備えている。少量のデータやバックアップ用のデータ、あるいは処理されて順次消えていくようなデータには通常のファイルが使われる。また、共有の必要があっても、構造化できないようなデータや流動的なデータは、グループウェアなど他の種類の情報システムで扱う方が適している。

情報システムのモデリングでは、情報をどのように管理するか、データ構造・保管期間・データの変換やそれらの形式をどのように設計するか、を考えることが重要である。たとえばデータベースの設計では、蓄積する情報の内容やデータの構造を決めている。これを概念データモデルとよぶ。DBMS は、このモデルをコンピュータ上に実現し、正しく運営・管理するためのソフトウェアであると考えることができる。

データベースは、データの意味を記述するメタデータ部分と対象となるデータそのものから構成される。新聞記事データベースを例に取れば、記事の見出しや掲載年月日はメタデータ部分であり、収録されている新聞記事の全文がデータそのものである。絵画データベースであれば、画題・製作者・制作年・大きさ・説明などがメタデータ部分であり、絵画の画像そのものがデータ部分である。

DBMS が提供する機能としては、次のものが挙げられる:

- 情報を問合せるための検索機能
- データベース内のデータの整合性を正しく保つための保全機能
- 複数ユーザが同時にアクセスできるようにする機能
- データの二重更新を防止するための排他的制御機能
- データ障害時に回復するための復元機能
- アクセス権を制御し機密を保護するためのセキュリティ機能

[1] 単に目的を定めてデータを集積したものを「データベース」と呼ぶ場合もある (広義のデータベース)。

また、情報を効率的に利用するための仕掛け、協調作業を容易にするための仕掛け、情報発信のための仕掛けなども備わっている。

データベースの枠組みとその選択

DBMS にはいくつかの枠組み (データモデル) がある。現在もっとも普及しているのは、データを複数の「表」の集まりとして表現する、**リレーショナルデータベース** (RDB、Relational DataBase) と呼ばれる種類のものである。

今日 RDB が主流である理由としては、理論的土台があり、これに基づいてデータの検索や加工が一括して (簡潔に) 指定可能であること、そのため構造が解かり易く業務系や分析系のアプリケーションに適していること、標準化が進んでいてベンダー (製造業者) が違ってもほぼ同一の方法で利用できることが挙げられる。

別の枠組みとして、個々のデータやその関連を RDB より自由に定義可能な**オブジェクト指向データベース**がある。オブジェクト指向データベースはデータ表現の柔軟性は高いが、現時点ではあまり普及していない。これは、RDB のような一括した検索指定が行いにくいため分析系のアプリケーションには不向きであること、リレーショナルデータベースと比べて標準化が進んでいないことなどが挙げられる。

しかしマルチメディア系や業務系のアプリケーションには適しているため、リレーショナルデータベースでは機能や性能面で満足できないような場合にはオブジェクト指向データベースの適用が検討される。

情報システムの設計では、情報を管理するために、システムの全体を考えて利用するデータベースの枠組みを選択することも重要な仕事である。(コラム 7 参照)。

8.4.3　情報の活用

例: ニュースソースの活用

人間は行動する前に情報を取得し、意思決定をしている。ここでは、情報の取得先にあるしくみに目を向けてみよう。

一日のはじまりにテレビのニュースを見て情報行動の意思決定をしている人が多い。たとえば、テレビが発している時刻という情報を利用し

(コラム 7)
データベースの誕生と推移
人間は、何らかの意思決定に必要な情報を探し集め、それを利用している。必要になってから収集を始めるばかりでなく、普段から情報活動に必要になるだろう情報を整理しておくこともある。
もともとデータベースは、図書館の文献データを編集処理した際の副産物として、それを磁気テープに収録したのが始まりであった。1960 年代のことである。それが文献データベースと呼ばれるようになり、その後、バッチ処理で行なう外部への情報提供サービスへと発展した。
1970-1980 年代にかけては、大型コンピュータと公衆通信回線を利用したオンライン検索サービスが生れ進化している。この間、CD-ROM にデータベースを掲載して提供するサービスも行われたが、容量不足と操作の複雑さなどの理由からあまり活用されないまま役割を終えてしまった。
1990 年代後半になるとインターネットが普及し、さらに WWW が出現したことで、データベースによる情報サービスの主流もインターネットを媒体とするものに移行している。
企業のデータベースではもともと、組織の活動に関係する全てのデータをまとめて管理し利用することが考えられた。そこで、企業のどの部門からでも情報を利用できるようにするため、データを一つのコンピュータに集めて管理する集中管理方式が一般的であった。しかし、情報システムの利用技術が向上しタイムリーな処理の必要性が問われるようになって、次第に分散管理へと移行している。

ている人がいる。その日の行動に影響するかもしれない出来事をチェックして、トラブル回避に役立てている人がいる。天気予報を見て、雨具が必要か否かを判断している人がいる。

このような情報活用は、いずれも次の行為や行動への意思決定に必要なことである。テレビの視聴者がこのような情報収集活動を日常的にできる背後には、番組作成の情報システムがあり、そこで使用する情報源の維持と管理が時々刻々となされている。その情報源は、さまざまな組織が採取し加工し蓄積したデータである。

たとえば天気予報という番組に注目すると、日本の気象情報を担当する中央官庁である気象庁が構築している仕掛けが見えてくる。気象資料自動編集中継システムでは、国内ばかりでなく世界各国と気象観測データの交換をしているし、数値解析予報システムでは大気現象を地球規模でとらえて解析している。これらのしくみによって、正確な天気予報や防災情報が伝達されるのである。

国内では、観測データ収集のために、全国規模で情報交換ができる次のような階層的なネットワークが構成されている:

- アメダス (気象庁地域気象観測システム) センター
- 気象衛星センター
- 地震活動監視機関
- 防災センター

天気予報の作成には常に最新で膨大な気象観測データが必要であり、いつ発生するかわからない防災情報を間違いなく迅速に中継するためにはリアルタイム性と高い信頼性が求められる。このような情報を 365 日 24 時間ノンストップで発信するために、システムを二重化し、運用中にトラブルが発生したら自動的に切り替えられるようにしていることによって、運用の連続性が確保されている。また、短時間に多量に発生するトランザクション (単位処理) を高速に処理するため、データベースやネットワークなどが随所に取り入れられている。

例: 公共バスの利用と情報の活用

朝の情報収集のあと、通学や通勤のためにバスに乗るとしよう。バスの運行にも次のようなさまざまな情報が活用されている:

- 道路交通システムから送られてくる以下の情報
  - 道路交通情報
  - 安全運転のための危険警告
  - 交通事故時などの交通規制情報
  - 道路管理の効率化のための通行規則情報
- バス優先/ロケーションシステムが提供する公共交通利用情報
- 交差点や横断歩道の信号機
- 歩行者支援のための経路案内や危険防止案内

乗客に提供される情報としては、表示板で表示される乗車区間と次に止まる停留所名と料金、乗車時に発行される整理番号、降車通報のブザー音などがある。乗客は整理番号と料金表示を確認して運賃を支払う。

運賃支払にプリペイドカードを利用すれば、乗車情報が記録され運賃はカードの残高から引き落とされる。複数バス会社で共通に利用できるバス共通カードのシステムには、カード番号と乗車情報から会社ごとの決済ができる仕掛けがある。ここでも表示システム、通報システム、カード読取システムなどが機能している。

これらそれぞれのシステムで採取されるデータはデータベースに蓄積され関係者が共用できるようになっている。

索 引