情報処理 (3)

情報量・エディタ

増原 英彦

1　今週の理解目標

- 情報量とは何か
- 数値と文字の符号化方法
- エディタの使い方 (基本的な入力方法・切り貼り・検索と置換)

2　情報の量と表現

「情報」とは量ることのできる単位だろうか? 日常的には「この講義は情報が多い」「イスラム教に関しては日本語の情報が少ない」「百聞は一見にしかず」などと言うように、情報の量を比べている。計算機システムが関係する場面では「1 時間のビデオ情報は CD-R 1 枚に書き込めるか?」「デジタルカメラで写真をもう 1 枚撮るためには電子メールを何通消せばよいか?」といった比較も行われている。

このような「情報の量」を比べる手段として**情報量**という概念がある。ただし、この概念は理論的なものであり、情報の価値を比較しているものではなく、情報の中身の判断とは独立に保存したり伝えるような場合における量を決めている。(上の例の中には「受け取り手にとって有用かどうか」によって情報が多い・少ないと言っている場合もあることに注意せよ。)

おおまかには情報量は次のように定義する。

ある事実の情報量は、その事実が「何通りの可能性の中から 1 つが選ばれたか」で決める

例えば

- 「10 円玉が表を向いている」ことは「**2** 通り (表か裏か) から 1 つ選ばれた」
- 「サイコロの目が 3 である」ことは「**6** 通りから 1 つ選ばれた」
- 「8 面体サイコロの目が 1 である」ことは「**8** 通りから 1 つ選ばれた」
- 「10 桁の宝くじの当選番号が 1234567890 である」ことは「$\mathbf{10^{10}}$ 通りから 1 つ選ばれた」
- 「10 円玉と 100 円玉が表と裏を向いている」ことは「**4** 通りから 1 つ選ばれた」
- 「10 円玉と 100 円玉と 500 円玉が表と裏と表を向いている」ことは「**8** 通りから 1 つ選ばれた」

(注: ここでは n 通りの可能性は全て等しい確率で起きるものと仮定している。等確率でない場合の情報量は情報の圧縮を説明する際に扱う。)

この可能性の数を使って情報の量が比較できる。例えば「3 個のコインの表裏を使って表わせる情報」と「6 面体サイコロを使って表わせる情報」を比べると、前者が 8 通りから、また後者が 6 通りから選ばれている。従って、3 個のコインの表裏の方がより多い情報量を持っていることになる。

この情報量を数量化した定義は次のようなものである:

定義 (情報量)　n 通りの可能性から 1 つ選ばれた事実の情報量は $\log_2 n$ ビットとする。

この定義は、次のような性質がある。まず、「1 個のコインの表裏」のように「2 通りの可能性」から選ばれた情報は $\log_2 2 = 1$ と最小単位になる。(両面が「表」であるコインが表を向いているという事実は、1 通りの可能性から選ばれたことになるので、情報量は $\log_2 1 = 0$ となる。これも自然であろう。) また「3 個のコインの表裏」は 8 通りから選ばれるので $\log_2 8 = 3$ で「1 個のコインの表裏」の 3 倍の情報量となる。

この考察を一般化すると、n ビットの情報量と m ビットの情報量を組み合わせると、$n+m$ ビットの情報量になる。例えばコイン n 個の表裏を使うと 2^n 通りの可能性から 1 つの情報、つまり $\log_2 2^n = n$ ビットの情報が表わせる。

情報量の定義は、計算機内部の情報の表現 (**デジタル表現**) によく符合する。電子計算機の内部では、真空管・トランジスタなどのスイッチ素子を使って「回路に電流が流れているか・いないか」あるいは「回路に電圧がかかっているか・いないか」の「2 通りの可能性のどちらか」によって情報を表現しているためである。もちろん電流や電圧は連続的に変化させることのできる量であるから、「電圧を 8 段階に制御して、8 通りの可能性の 1 つ」といった表わし方も考えられる。しかし、電圧・電流を多段階に制御する回路よりも、2 段階の制御をする回路を複数組み合わせた方が有利であったため、現在のほぼ全ての電子計算機が「2 通りの可能性のどちらか」を基本的な表現としている。

計算機内部で表現する情報には数値・文字・画像・音声などがあるが、これらは全て範囲を決めておき、個々の情報は「n 通りの可能性の 1 つ」と考え、何ビットの情報の組み合わせによって表わす。

この複数のビットの組み合わせを簡単に書くために、1 ビットの情報は「0 か 1 のどちらか」であるとして説明をする。1 ビットの情報を n 個組み合わせた n ビットの情報は、0 か 1 を n 個並べて書くことにする。例えば、2 ビットであれば「00」「01」「10」「11」のように 4 通り、3 ビットであれば「000」「001」「010」「011」「100」「101」「110」「111」の 8 通りの場合がある。

以下では、数値と文字を計算機の内部で表現する方法について説明する。

3　数値の表現

3.1　2 進数

整数は 2 進数によって表わす。我々が普段使用しているのは、各桁に 0 から 9 までの 10 個の数を使う 10 進数であるが、2 進数は各桁に 0 と 1 の 2 個の数を使って数値を表わす方法である。従って 2 進数で表わされた数は「1011011」のように 0 と 1 だけを使って表わされた数となる。

n 桁の 2 進数を「$b_{n-1}b_{n-2}\ldots b_2b_1b_0$」のように書くことにする。(区別のために 2 進数には「」をつけることにする。) 各桁の b_i は「0」か「1」のどちらかである。この 2 進数が表わしている値は

$$b_{n-1}\times 2^{n-1} + b_{n-2}\times 2^{n-2} + \cdots + b_2\times 2^2 + b_1\times 2^1 + b_0\times 2^0 \left(= \sum_{i=0}^{n-1} b_i \times 2^i\right) \tag{1}$$

によって求められる。(この式は 10 進数で書かれていることに注意。) つまり、下から i 桁目が「1」であるということは、10 進数で 2 の i 乗ぶんの値があるということである。例えば「101」という 2 進数は 2 桁目と 0 桁目が「1」であるので、10 進数では $2^2 + 2^0 = 4 + 1 = 5$ という数を表わしている。「1011011」であれば 6,4,3,1,0 桁目が「1」であるので $2^6 + 2^4 + 2^3 + 2^1 + 2^0 = 64 + 16 + 8 + 2 + 1 = 91$ となる。

ある数 d を 2 進数で書き直す方法は $d = \sum_{i=0}^{n-1} b_i \times 2^i$ となるような数列 $\{b_i\}$ を求めることになるが、これは各桁について $b_i = (d \text{ div } 2^i) \bmod 2$ という計算によって求められる (div は商、mod は余りを求める演算で

ある)。例えば、91 の 2 進数表現は

$$\begin{aligned}
b_0 &= (91 \text{ div } 2^0) \bmod 2 = 91 \bmod 2 = 1\\
b_1 &= (91 \text{ div } 2^1) \bmod 2 = 45 \bmod 2 = 1\\
b_2 &= (91 \text{ div } 2^2) \bmod 2 = 22 \bmod 2 = 0\\
b_3 &= (91 \text{ div } 2^3) \bmod 2 = 11 \bmod 2 = 1\\
b_4 &= (91 \text{ div } 2^4) \bmod 2 = \ 5 \bmod 2 = 1\\
b_5 &= (91 \text{ div } 2^5) \bmod 2 = \ 2 \bmod 2 = 0\\
b_6 &= (91 \text{ div } 2^6) \bmod 2 = \ 1 \bmod 2 = 1\\
b_7 &= (91 \text{ div } 2^7) \bmod 2 = \ 0 \bmod 2 = 0
\end{aligned}$$

より「01011011」となる。

2 進数は 10 進数の各桁を 2 個の数に減らしただけなので、加減乗除の計算を含む数学的な法則はほとんどそのまま成り立つ。(2 進数だけでなく 3 進数,4 進数なども同じである。)

加算: 10 進数の加算は「1 番下の位から順に、各位の数どうし (と下の位からの繰り上がり) を足し、それを 10 で割った余りをその位の数、10 で割った商を繰り上がりとする」ことで計算できる。2 進数でも同様に「1 番下の位から順に、各位の数どうし (と下の位からの繰り上がり) を足し、それを 2 で割った余りをその位の数、2 で割った商を繰り上がりとする」ことで計算できる。

例えば 2 進数の「1011」と「0011」を足す場合には、次のように計算できる:

$$\begin{array}{rcccc}
 & 1 & 0 & 1 & 1\\
+) & 0 & 0 & 1 & 1\\
\hline
 & 1 & 1 & 1 & 0
\end{array}$$

1. まず 1 番下の位の「1」と「1」を足すと (10 進数で)2 になるので、1 番下が「0」となり「1」繰り上がる
2. 下から 2 番目の位の「1」と「1」と繰り上がりの「1」を足すと (10 進数で) 3 になるので、下から 2 番目は「1」となり「1」繰り上がる
3. 下から 3 番目の位の「0」と「0」と繰り上がりの「1」を足すと (10 進数で) 1 になるので、下から 3 番目は「1」となる
4. 下から 4 番目の位の「1」と「0」を足すと (10 進数で) 1 になるので、下から 4 番目は「1」となる

同様に引き算、掛け算、割り算なども 10 進数と同じ方法で行うことができる。

また、10 進数には「1 桁あがると 10 倍」という関係があるが、2 進数にも 1 桁あがると「10」倍 (つまり 2 倍) になるという関係がある。このことは以下のように示すことができる。

$$\begin{aligned}
「b_{n-1}b_{n-2}\ldots b_2b_1b_0」\times「10」 &= \left(\sum_{i=0}^{n-1} b_i \times 2^i\right) \times 2 = \sum_{i=0}^{n-1}(b_i \times 2^{i+1})\\
&= 0 \times 2^0 + \sum_{i=1}^{n} b_{i-1} \times 2^i = 「b_{n-1}b_{n-2}\ldots b_2b_1b_00」
\end{aligned}$$

問題 3–1: (2 進数と 10 進数) 次の 2 進数を 10 進数に直せ。

- 「111」
- 「1001」
- 「1100」
- 「10101」
- 「100001」
- 「1000010」

問題 3–2: (2 進数の足し算) 次のような 2 進数どうしの合計を 2 進数で求めよ。足される 2 進数と足された 2 進数をそれぞれ 10 進数に直し、10 進数でも正しい足し算になっているかを確認せよ。

- 「1001」+「1100」
- 「1100」+「10101」
- 「100001」+「100001」

計算機で数値を表現する方法には正の整数、符号付き整数、固定小数点数、浮動小数点数の 4 種類がある。それぞれ表わせる値の範囲が違うので、表わす内容に最もふさわしい表現方法を使うことになっている。

3.2 正の整数

現在のパーソナルコンピュータの CPU は、32 桁の 2 進数を正の整数として表現し、計算することができる。つまり「00000000000000000000000000000000」から「11111111111111111111111111111111」($= \sum_{i=0}^{31} 2^i = 4,294,967,295$) までの数を扱うことができる。

約 43 億までの数を表わせるわけだが、これよりも大きな数を扱いたい場合もある。例えば国家や企業の会計であれば、これ以上の数を扱えなければいけない。そのような場合は、表わしたい数を 2^{32} で割った商と余りの 2 つの数をそれぞれ 32 桁の 2 進数で表わす。これで 64 桁の 2 進数 (10 進数で約 1800 京) までの整数を表わすことができる。このような数を計算するためには、商部分と余り部分をそれぞれ別に計算することになる。

より高価な計算機では CPU が 64 桁の 2 進数を直接扱えるものもある。一方、家電製品などに内蔵されている CPU では 8 桁の 2 進数、つまり 0 から 255 までの数しか直接扱うことのできないものもある。

3.3 符号付整数

負の整数を含めた整数 (**符号付整数**と言われる) を表わすためには、「その数が正か負か」という情報を含めなければならないため、正だけの整数とくらべて 1 ビット多くの情報が必要となる。現在のパーソナルコンピュータの CPU は 2 進数 31 桁の符号付整数を扱うことができる。つまり、符号に関する情報もあわせるとちょうど 32 ビットの情報量の数となる。

符号付整数は**2 の補数表現**によって表わされる。補数表現を説明するために「10 進数の 10 の補数表現」を先に紹介する。2 の補数表現は同じことを 2 進数で行っただけである。

いま、0 から 9 までの数を 3 桁使って符号つきの 10 進数を 10 の補数表現によって表わすことを考える。(マイナス記号は使わない。) $0, 1, 2, 3, \ldots$ のような正の数はそのまま「000」,「001」,「002」,「003」, ... のように表わす。一方、負の数 $-1, -2, -3, \ldots$ は 1000 を加えた数を使って「999」,「998」,「997」, ... のように表わす。つまり -1 に 1000 を加えると 999 なので「999」と表わすといった具合である。この対応をまとめると下のようになる:

もとの数	-500	-499	$\cdots$	-2	-1	0	1	2	$\cdots$	499
10 の補数表現	「500」	「501」	$\cdots$	「998」	「999」	「0」	「1」	「2」	$\cdots$	「499」

この表現方法の特徴は、四則演算の結果が保たれる点である。つまり、-499 のような負の数の 10 の補数表現「501」を正の整数だと思って計算をして、その結果を 10 の補数表現だと思ってもとの数に戻しても結果が正しいということである。例えば、-499 と 2 を足す場合を考えてみると、これらの数の 10 の補数表現は「501」,「2」である。これらを正の整数だと思って足すと「503」になる。これを 10 の補数表現だと思ってもとの数に戻すと -497 になっているが、確かにこれは -499 と 2 の和になっている。(下図)

もとの数	-499	$+$	2	$\rightarrow$	-497
	$\downarrow$		$\downarrow$		$\uparrow$
10 の補数表現	「501」	$+$	「2」	$\rightarrow$	「503」

同様に、$1 + (-2)$ は「1」+「998」=「999」となり、これは -1 に対応している。また、結果が 3 桁以上になる場合は、下 3 桁だけを残せばよい。例えば $2 + (-1)$ は「2」+「999」=「1001」となるが、下 3 桁だけの「001」は 1 に対応している。足し算だけでなく、掛け算も同様である。例えば -2×2 は「998」×「2」=「1996」となり、下 3 桁の「996」は -4 に対応している。

n ビットの 2 の補数表現は、負の数に 2^n を加えた数を使って表わす。例えば 4 ビットの場合、-1 から -8 までの数は $2^4 = 16$ を加えた正の数を 2 進数で表わす。この対応をまとめると下のようになる:

もとの数	-8	-7	$\cdots$	-2	-1	0	1	2	$\cdots$	7
2 の補数表現	「1000」	「1001」	$\cdots$	「1110」	「1111」	「0000」	「0001」	「0010」	$\cdots$	「0111」

この場合もやはり、2 の補数表現によって表わされた数を、**??**で示したような正の整数の表現だと思って計算をしても正しく計算ができる。例えば $-7+2 \to$「1001」+「0010」=「1011」$\to -5$ といった具合である。

2 の補数表現を使うことで、CPU の内部では整数が「正だけの整数」か「符号付整数」のどちらの場合であっても、ほとんど同じ回路を使って計算をすることができるため、ハードウェアの簡略化ができる。

問題 3–3: (負数の表現) 符号付きの整数を表わす別の方法として、「一番上位の桁が 0 なら正、1 なら負」とする方法もある。例えば 4 ビットであれば「0101」は正の「101」(つまり 5) を「1101」は負の「101」(つまり -5) を表わしている、といったものである。このような表現をとったときに、「0101」「1101」などの表現を「4 ビットの正の整数」だと思って計算をしたら結果が変わってしまうことを確めよ。

3.4 実数

小数も含めた数を表現する方法の 1 つは、**固定小数点数**によるものである。これは、小数点以下の桁数を予め決めておく方法である。例えばドル通貨で金額を表わす際には、小数点以下 2 桁 (つまりセントの単位) まであれば充分である。

このような数は、定数倍した整数によって表現する。例えばドル通貨であれば常に 100 倍した数 (つまりセントに換算した額) を使うようなものである。計算機では整数を 2 進数によって表わしているので、2^n 倍した整数によって表わすことになる。例えば、$2^8 = 256$ 倍した整数を使えば、1/256 の精度で小数点以下の大きさも表現ができる。固定小数点数の計算は、整数と同じようにできる (ただし、掛け算や割り算において桁数を補正する必要がある)。

小数のもう一つの表現方法は、**浮動小数点数**によるものである。これは、科学・工学分野で使わている仮数と指数による 2.9979×10^8 のような表現を 2 進数で行っているものである。例えば、2.9979×10^8 という 10 進数は

$$「1.\underline{00011101111001101110110}」\times 2^{「\underline{10011011}」-127}$$

であるので、符号、指数部、仮数部を並べて「01001101100011101111001101110110」のように表現する。(仮数部は小数を 2 進数で書いている。従って「0.1」とは 10 進数で $2^{-1} = 0.5$ である。)

4 文字の表現

文字情報は、1 つ 1 つの文字を整数に割り当てて表現する。例えば、「A」は 65、「D」は 68、「z」は 122 といった具合である。文字の表現において面倒な問題は、言語によって使われる文字が異なり、また、言語によっては多数の文字を使っている点である。

これらの点を明確に議論するために用語を整理しておく:

- **文字集合**とは、文字を集めたまとまりのことである。例えば英語の文章を表現することを考えているのであれば、アルファベットの大文字/小文字・数字・句読点などの記号などが含まれた文字集合を考えればよい。日本字の集合は、ひらがな・カタカナ・漢字の集まりになる。漢字にはまた、常用漢字・人名用漢字のような文字集合を考えることができる。

- **文字コード**とは、ある文字集合に含まれている文字 1 つ 1 つに割り当てられた番号のことである。このコードは文字集合ごとに決めることができる。従って例えば、日本字集合の 1234 番と中国字集合の 1234 番は異なる字を表わすということが起こり得る。

- **符号化方式**とは、文字コードをビット列によって表わす規則のことである。文字集合を 1 つしか使わない単純な場合であれば、文字コードを整数と同様に数ビットで表わすだけである。複数の文字集合を使う場合には、符号化されたビット列から「どの文字集合」の「何番の文字」という情報が分からなければならないため、より複雑な方式がとられる。

4.1 欧米言語の文字の符号化

英語のための文字集合は 26 文字のアルファベットである。実際には、大文字・小文字・数字・句読点などを含めて 100 弱の文字からなる文字集合を使うことが一般的である。これに番号を割り当てる方法の 1 つが**ASCII** と呼ばれる米国の規格であり (ISO 646 という国際規格にもなっている)、各文字に 0 から 127 の番号 (コード) を割り当てている。このコードは 7 ビットで表現可能なので、そのまま 7 ビットまたは 8 ビットの整数として符号化する。例えば「A brown fox」という文を ASCII コードによって符号化すると下のようになる (ここでは符号化されたものを 10 進数で書き直している)。空白や読点にもコードがあることに注意せよ。

ビット列 (10 進数)	65	32	98	114	111	119	110	32	102	111	120	46
文字	A		b	r	o	w	n		f	o	x	.

英語以外のヨーロッパ言語には、英語と同じアルファベットに加えてÁ, Â, Ä, Ø, ß, ç のような文字も使われる。また、ロシア語やギリシャ語のように英語と違うアルファベットを使う言語もある。このような文字集合のコードを定めている規格には ISO 8859-1 などがある。この規格では、これらの文字に 160 から 255 の番号 (コード) を割り当てている。このコードは ASCII コードと重ならないように作ってあるので、ASCII コードと ISO 8859-1 コードをまとめて 8 ビットの整数として符号化することも行われている。(ISO 8859-1 は西欧諸国の言語の文字集合を定めている。他に東欧の言語、スカンジナビア・バルト諸国の言語、キリル文字、アラビア文字等々は ISO 8859-2, ISO 8859-3 などと別々に定められている。)

4.2 日本字の符号化

日本では日常的に数千の漢字が使われている。これを表わすための規格の 1 つが JIS X 0208 と呼ばれる文字コードで、約 6 千文字の文字集合に対する番号 (コード) を定めている。この文字コードは、1 つの文字に 1〜94 の区コードと 1–94 の点コードの 2 つのコードの組み合わせを割り当てている。例えば「情」という文字には “30 区 80 点” というコードが割り当てられている、といった具合である。

この日本字の文字コードを符号化する方法は複数ある。欧米言語の文字コードの場合、符号化は (事実上) 一通りしかなかったため「文字コード」と「その符号化」の区別を意識する必要がなかったが、日本字の場合は複数の符号化方式があるゆえ区別しなければいけない。また、それぞれの符号化方式は ASCII コードと一緒に使えるための工夫のために複雑なものになっている。この工夫とは、日本字も使えるソフトウェアで英語文字集合だけを使った場合に、英語向けのソフトウェアと同じ符号化を行うというものである。例えば、日本人が英文のメールを書いて米国人に送った場合、ASCII コードで符号化して送られた方が好都合であろう。

以下では、JIS コードを符号化する 3 つの方式を紹介する。

- **JIS コード**は文字コードだけでなく、符号化方式も定めている。JIS コードの定める符号化方式は、**文字集合を切り替える**というものである。

  例えば「abc 情報 ef」という文があったときに、JIS コードでは「97 98 99 27 36 66 62 112 74 115 27 40 66 101 102」といった 7 ビットの整数に符号化する。これは、「27 36 66」という 3 つの整数の並びに「JIS X 208 コードへ切り替える」という意味を持たせているので、それ以降は区番号と点番号 (に 32 を足した数) が並んでいることになる。

97	98	99	27	36	66	62	112	74	115	27	40	66	101	102
a	b	c	日本字に切替			情		報		ASCII に切替			e	f

  この方式は、文字コードが重複するような文字集合どうしであっても文字集合の切り替えによって区別できるため、様々な文字集合を混在させることができる。

  一方で切り替えためのビット列が現われるので、複数の文字集合が入り乱れた文章を符号化した場合、他の符号化方式よりも長い表現になる。

- **EUC** は、ASCII 文字集合と日本字 (あるいは中国語の文字・ハングル) の 2 つの文字集合を混在させるための符号化方式である。ASCII 文字集合は、そのまま ASCII コードを 8 ビット整数として表わし、JIS X 208 文字集合 (漢字) は、区番号・点番号に 160 を加えた 8 ビット整数を並べる。

  例えば「abc 情報 ef」は以下のように符号化される。ここで「情」は 30 区 80 点であったので、160 ずつ足された 190,240 という整数に符号化されている。

97	98	99	190	240	202	243	101	102
a	b	c	情		報		e	f

  JIS コードと違い、文字集合の切り替えがないので、短く符号化ができる。一方で、日本語と中国語を混在させるようなことはできない。

- **シフト JIS** は、やはり ASCII 文字集合と日本字の 2 つの文字集合を混在させるための符号化方式である。ASCII 文字集合は、そのまま ASCII コードを 8 ビット整数として表わし、JIS X 208 文字集合 (漢字) は、区番号を 129 から 159 および 224 から 252 の整数に、点番号を 64 から 126 および 128 から 252 の整数に符号化して表わす。

97	98	99	143	238	149	241	101	102
a	b	c	情		報		e	f

4.3 多国語の文字の符号化方式

Unicode は世界の主要な言語の文字集合を 1 つにまとめて 0 から 65535 の番号 (コード) を割り当てた文字コードである。ASCII 文字集合に含まれる文字を 8 ビット整数で表わし、他の文字を 16,24,32 ビット整数で表わす UTF-8 と呼ばれる符号化と、16 ビット整数で表わす UTF-16 と呼ばれる符号化方式がある。(下図)

UTF-8:

97	98	99	346	203	205	345	240	261	101	102
a	b	c	情			報			e	f

UTF-16:

254	255	0	97	0	98	0	99	140	96	197	88	0	101	0	102
(空)		a		b		c		情		報		e		f	

文字集合を 1 つにまとめているため、JIS コードのような文字集合の切り替えを行わなくても複数の言語の文字を混在させることができる。一方で、中国語や日本語のような文字数の多い言語の文字も 0 から 65535 の番号の中に含めるために、各国の言語で「同じ」形をした字に同じコードを割り当てることを行っている。そのため、複数の言語が混在している文章では、各文字がどの言語で書かれた文章に含まれていたかを区別することは、Unicode 単独ではできない。

5 実習: エディタ

練習 3–4: (エディタの基本操作) HWB の「15. エディタ」を読み、エディタの基本操作を習得せよ。HWB では「テキストエディット (TextEdit)」と「Emacs」という 2 種類のエディタが紹介されているが「情報処理」の範囲ではどちらでも同じようなものであるので、どちらか一方だけを読めばよい。今後の実習を続けるためには、最低でも以下のような操作が必要となる:

- ファイルを開く・ファイルへ保存・文字の挿入・削除
- 文章の切り貼り・検索・置換
- 1 つのファイルの文章を (全て) 別のファイルに写す

練習 3–5: (日本語の編集) 以下のいずれかの題で 400 字程度の文章を書き、「sakubun.txt」というファイル名で保存せよ。

1. 匿名掲示板の存在意義
2. マスメディアと Web 日記
3. 情報化社会の落とし穴
4. インターネットと国際化
5. 携帯電話の利用マナー

練習 3–6: (レポートの見本を作る) テキスト形式のレポートをエディタで作成して、電子メール等によって提出する機会は、これから多くなってくる。ところでレポートには、授業名、課題名、名前、日付けなど多くの情報を書かなければいけないが、これらを全て忘れずに、見栄えのするように配置することは神経と手間を使う作業である。

そこで、テキスト形式でレポートを作成する際に便利な「見本レポート」を作り、「report-mihon.txt」というファイル名で保存せよ。目的は、実際にレポートを書く際には、この「見本レポート」の複製を作り、その中にレポート個有の情報 (課題名や日付、そして内容) を埋めてゆくことで体裁の整ったレポートを完成させるというものである。

下は「見本レポート」の一例であるが、この通りに作る必要はない。(むしろ、これと違う体裁を考えるべきである。) 実際、この「見本」に加えて課題の番号、授業を担当している教師の名前、履習している曜日と時限、自分の科類クラス学生証番号などがあった方がよいかも知れない。また、本文の章立てをより詳しくしておいたり、章の切れ目が分かりやすいように記号などを使って目立たせるといった工夫の余地は沢山ある。

```
                【授業名】レポート「【課題名】」
                                【年】年【月】月【日】日
                                        情報 一郎

1．課題の概略

【ここに課題の概略を書く】

…

4．結果

【ここに結果を書く】

5．考察

【ここに考察を書く】
```

練習 3–7: (レポート見本の利用) 練習 3–5 で書いた文章と、練習 3–6 で作ったレポートの見本をあわせて、1 つのレポートを作成し、「report1.txt」というファイル名で保存せよ。このとき、練習 3–5 と練習 3–6 で作った元々のファイルはそのままにして、新しいファイルにレポートを完成させるように注意せよ。

(ここで作成したレポートは、次回以降に実際に提出する。)