C言語によるプログラミング 2:
変数の型、標準入出力

A Pedestrian Approach
to
the C Programming Language

目 次

図 目 次

表 目 次

2　変数の型、標準入出力

2.1　変数の型、およびかんたんな演算

2.1.1　かんたんな整数演算

かんたんな整数演算１

```
/*  sansu1.c    */

#include <stdio.h>

main()
{
    int a, b, c; /* 変数の型宣言。a, b, c は整数型の変数。*/

    a = 200;       /* 変数の初期化。a の箱に 200 が入る。*/
    b = 1300;      /* 変数の初期化。b の箱に 200 が入る。*/

    c = a + b;     /* a の箱の中味と b の箱の中味をたして c の箱に入れる。*/

    printf("a + b = %d\n", c); /* c の箱の中味を 10 進整数として端末に印字する。*/
}
```

結果

```
$ g++ sansu1.c
$ ./a.out
a  + b = 1500
```

ここでは、200 ＋ 1300 = 1500 をプログラムをつかっておこなっている。じっさいにどういう過程がコンピュータのなかでおこなわれているかの詳細は、たとえば、[9] を参照されたい。簡単な説明は、ソースコードのコメント、およびこの後の記述のとおりである。

◇ 変数、その型宣言、および初期化 [10, 11]

変数というのは、メモリ内の位置につけられた名前のことである。ホテルや旅館の部屋の名前、あるいは部屋番号とおもえばよい。この部屋に、プログラムにしたがって、ビット (0 または 1) の列が格納されることになる。これが初期化である。型宣言は、この部屋のタイプを指定する。

ここでは、ソースプログラムのなかで初期化がおこなわれている。対話的に、プログラムを走らせてから、変数を初期化する方法は、次の例を参照されたい。

つぎに、さらにくわしく見ていくことにする。

●変数の型宣言

C では、つかう変数の型をすべてプログラムの最初に (厳密にはその変数を使用する有効範囲の最初

に) 宣言しておく必要がある。変数につかえるのは、英数字(もちろん半角英数である。アンダースコア「_」も英字に含む) である。また、初めの１文字は英字 である必要がある。C では、大文字と小文字を区別する（Linux も然り)。変数名のつけかたは、各 OS で習慣があるようである [12]。しかし、基本的に小文字をつかうのは共通している。UNIX の場合、変数名をすべて小文字にする傾向がある。記号定数には大文字をつかう。

例

整数型　`int a;`

　　`int x_max;`

単精度実数型　`float volume_sphere;`

●変数の初期化

```
a = 200;
```

この部分は、先につくった a というメモリ内の場所 (仮想的な箱のようなもの) にビット (0 または 1) の列を入れている。C での「= 」は数学でつかう左辺と右辺の等価を意味するものではなく、「右辺の値を左辺の値に代入」する『代入演算子』であることに注意されたい。（※　論理的な等価は、[13] の § 3.3 にあるように、「==」で表わす。また、言語によっては、「=」が論理的な等価を意味するものもある。）

いまの場合、a の箱に 200 がビットの列で入る。すなわち、2 進数で入ることになる。

　したがって、奇妙に思うかもしれないが

```
a = a + 1;
```

という記述も可能である。（これは何をしているのか？）

　元に戻ると

```
b = 1300;
```

の部分もまた同様の「代入」操作である。

●演算過程

```
c = a + b;
```

ここでは、CPU(中央処理装置) がメモリー内の a, b の箱からビットの列を取り出し、データ・レジスタ (CPU 内のメモリ) に置く。つぎにこのレジスタ内のふたつのデータ (いまの場合、200 と 1300) を演算部に送り、さらに、両者を加える。結果は一時的に演算部のレジスタに貯えられ、つぎにメモリー内の c の箱に転送される。

　いやー、CPU さん、御苦労なことです。くわしくは、[9] の 5 章を参照のこと。

●標準出力

端末に書式指定した仕様にしたがって、変数を出力する (標準出力)。この関数は、標準ライブラリーのなかの入出力にかんする関数群を収めるライブラリー`<stdio.h>`に入っている。これをつかえばよい。

例

```
printf("a + b = %d\n", c);
```

変数 c を、`%d` の箇所に、10 進数で表示して出力する。この変数 c の内容は、前記の演算過程で転送されてきたものである。

●コメント

`/* 変数の型宣言。a, b, c は整数型の変数。*/`

のように、/* と*/ でかこまれた部分は、ソースコードとしては、無視される。この間にプログラムや変数の説明をする。これが コメント である。コメントをていねいに書くと後から見たときにプログラムがわかりやすくなる。また、他人にとってもそのプログラムはわかりやすい。この部分には日本語表記してもよい。ただし、「/」および「*」自体は、半角英数でなければならない。

かんたんな整数演算 2

```
/* sansu2.c */

#include <stdio.h>

main()
{
    int a, b, c;          /* 変数の型宣言。a, b, c は整数型の変数。*/

    printf("整数 a をあたえてください。a: ");
    scanf("%d", &a);        /* キーボードから変数 a を 10 進整数で読み込む。 */
    printf("整数 b をあたえてください。b: ");
    scanf("%d", &b);        /* キーボードから変数 b を 10 進整数で読み込む   */

    c = a + b;            /* a の箱の中味と b の箱の中味をたして c の箱に入れる。*/
    printf("a + b = %d\n", c);
       /* c の箱の中味を 10 進整数として端末に印字する。*/

    c = a - b;    /* a の箱の中味から b の箱の中味を引いた結果を c の箱に入れる。*/
    printf("a - b = %d\n", c);
       /* c の箱の中味を 10 進整数として端末に印字する。*/

    c = a * b;/* a の箱の中味と b の箱の中味を掛け合わせた結果を c の箱に入れる。*/
    printf("a * b = %d\n", c);
       /* c の箱の中味を 10 進整数として端末に印字する。*/

    c = a / b;  /* a の箱の中味を b の箱の中味で割った結果 (商) を c の箱に入れる。*/
    printf("a / b = %d\n", c);
       /* c の箱の中味を 10 進整数として端末に印字する。*/
}
```

結果例

```
$ gcc sansu2.c
$ ./a.out
整数 a をあたえてください。a: 1000
整数 b をあたえてください。b: 20
a + b = 1020
```

```
a - b = 980
a * b = 20000
a / b = 50
$ ./a.out
整数 a をあたえてください。a: 100
整数 b をあたえてください。b: 30
a + b = 130
a - b = 70
a * b = 3000
a / b = 3
```

●標準入力

キーボードから書式にしたがって、変数の中味を読み込み、変数を初期化する (標準入力)。これによって、対話的に数値や文字を外部からあたえることができる。

いまの場合、プログラムが走ってから、あなたがキーボードから整数 a, b をあたえることになる。これは、ゲームで、プレーヤーの反応にしたがってつぎの展開が生じるのとおなじである。

●標準入力を促す標準出力での指示

したがって、プログラムをつくる方は、プレーヤーに (プログラムを実行する人) にどう反応すればよいか、を指示する必要がある。それは、さきほどの「標準出力」、すなわち printf による端末への印字によっておこなう。「整数 a をあたえてください。a:」等の部分がそれである。この実習の場合、プログラマーとプレーヤーは、基本的におなじである。それは、他でもない、あなた自身である。だが、本来、両者は別である。プレーヤーにたいする心配りがたいせつである。

例

```
scanf("%d", &a);
```

キーボードから変数 a を 10 進整数で読み込む。

※ 整数型変数どうしの演算では、割り算の結果は商を表わしていることに注意。

♣ 演習問題 1 ♣

1) 整数の組 (a, b) をあたえて、(b, c) を出力せよ。ただし、$c = a + b$ とする。

2) 整数 a をあたえて、$a^2 + a + 1$ を出力せよ。

2.1.2 コンピュータの基本構成から観た、プログラムとその実行

さて、プログラムで何をやっているかを、さらにふかく理解するために、コンピュータの基本構成から観たプログラムとその実行を先の「sansu2.c」を例にとってかんがえてみよう。

コンピュータの物理的な構成の概略

その前にまず、基本的な事項の復習である。

コンピュータの物理的な構成の概略は、つぎのような図 1 であらわすことができる [?]。

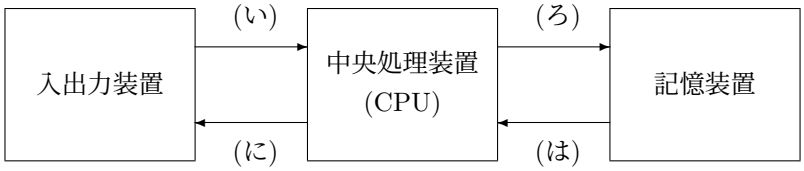

図 1: コンピュータの物理的な構成要素の概略

ここで、各装置の働きはつぎのようなものである。
・中央処理装置 (Central Processing Unit): 基本的な演算を実行する。
・入出力装置: キーボード、モニター、マウス、スキャナー、ケーブル等。これらを通して、プログラムやデータを記憶装置に格納したり、計算結果のやり取りをしたりしている。
・記憶装置: 主メモリ、補助メモリ (HDD, SSD, フラッシュメモリ等)。プログラムとデータが保持されている。

「sansu2.c」の場合、プログラムとそれに対応するデータの流れ

上記の復習の内容を援用すると、「sansu2.c」の場合、プログラムとそれに対応するデータの流れ【図 1 の (い)~(ろ)】を表にするとつぎのようになる。もちろん、CPU がプログラムが逐次読み込んで処理を実行するので、各過程の初めにデータは「(は) の矢印」を通過する。その後の各過程でのデータの動きを以下にしめしているのである。

プログラム	命令、データの流れ【図 1 の (い)~(ろ)】
`int a, b, c;`	(に)。 記憶装置に 4 バイトの箱 (領域) を確保する。
`printf...`	(は), (に)
`scanf...`	(い), (ろ)。a, b の箱にビットの列が入る (初期化)。
`a + b`	(は) , CPU。 a, b の箱からビットの列を取り出し、加算する。演算過程。データの加工。
`c = a + b;`	(は) , CPU の後、(ろ)。 結果を記憶装置の c の箱に格納する。
`printf("a + b = %d", c);`	(は), (に)。モニターに出力。
`a - b`	(は) , CPU。a, b の箱からビットの列を取り出し、減算する。演算過程。データの加工。
`c = a - b;`	(は) , CPU の後、(ろ)。 結果を記憶装置の c の箱に格納する。
`printf("a - b = %d", c);`	(は), (に)。モニターに出力。
...	...

表 1: プログラムとデータの流れ

情報処理の性質からの分類

さらに、情報処理の性質から分類すると、コンピュータをつかった情報処理には、(i) 情報の加工、(ii) 情報の蓄積・検索、(iii) 情報の伝達、がある [**?**]。

これに対応するデータの処理と先のコンピュータの物理的構成 (1) との関係はつぎのとおりである。(CPU はつねにデータの処理に関連している。)

	データの処理の種類	関連する主な物理的装置	データの流れ【図 1 の (い)~(ろ)】
(i)	データの加工	中央処理装置 (CPU)	(い)、 (は)
(ii)	データの蓄積・検索	記憶装置	(い)、(ろ)。および (は)
(iii)	データの伝達	入出力装置	(い)、(に)

表 2: データの処理の種類とデータの流れ

2.1.3 実数演算

例

「底面の半径 、および高さ h をあたえて、円錐の体積をもとめる」

を例にとって、変数の型と標準入出力をみてみよう。

プログラムをかくと、たとえば、つぎのようになる。

```
 /* cone1.c */

#include <stdio.h>

#define   PI   3.14f   /*ソースコードに PI が現われると実数 3.14 でおきかえる。 */

main()
{
   float   hankei, takasa, taiseki;  /* 変数の型宣言。hankei, takasa, taiseki は実数型。*/

   printf("底面の半径 を入力してください: ");
   scanf("%f", &hankei);   /* キーボードから入力した実数を変数 r 専用の箱に入れる。*/
   printf("円錐の高さを入力してください: ");
   scanf("%f", &takasa);      /* さきの scanf とおなじく標準入力。*/

   taiseki = (1.0 / 3.0) * PI * hankei * hankei * takasa ;   /* 円錐の体積を求める。
*/
   printf("円錐の体積は%5.2f です。\n", taiseki);     /*    標準出力。*/
}
```

◎注意◎

taiseki = (1.0 / 3.0) * PI * hankei * hankei * takasa ;

を
taiseki = 1 / 3 * PI * hankei * hankei * takasa ;
としたらどうなるか？

●実数の標準入出力 [15]

`printf("円錐の体積は%5.2f です。\n", taiseki);`
変数 taiseki を`%5.2f` の箇所に実数として、印字する。`%f` は実数を出力する書式指定。さらにここでは、`%5.2f` として、すくなくとも 5 文字幅、小数点以下 2 桁で表示 するように指定している。

♣ 演習問題 2♣

1) 四則演算を、実数変数をもちいておこなえ。
2) 半径 r の球の体積をもとめよ。(r はキーボードからあたえる)
3) 半径 r の球の表面積をもとめよ。(r はキーボードからあたえる)
4) 底面の一辺 a, 高さ h の正四角錐の体積をもとめよ。
(a, h はキーボードからあたえる。)

2.1.4 文字変数のあつかい

簡単のために、ここでは 1 バイト文字 (英数文字) をとりあつかう。

```
/*  ascii1.c */

#include <stdio.h>

main()
{
    char letter_a, letter_b;   /* 変数の型宣言。letter_a, letter_b は文字型の
変数。*/
    letter_a = 'a'; /* 変数の  初期化。letter_a の箱に文字 a が入る。  */

    printf("英数文字をひとつあたえてください。 ");
    scanf("%c", &letter_b);   /* キーボードから変数 a を英数文字として読み込む。
*/

    printf("ソースファイルにもともとあった文字は%c です。\n", letter_a);
    printf("あなたの入力した文字は%c です。\n", letter_b);
}
```

変数の宣言等は、整数のときの `int`、(単精度) 実数のときの `float` にたいして、`char` となっている。また、初期化に際しては、文字そのものが認識されるのではなく、その文字に対応する番号 (code) のビットの列が入ることになる。§ 2.3、とくに § 2.3.6 を参照のこと。他の違いは、書式指定が`%c` となっていることである。

2.2　変数の型宣言と標準入出力における書式指定のまとめ

●変数の型宣言

- 整数型 (例 `int a;`)
- 単精度実数型 (例 `float hankei;`)
- 倍精度実数型 (例 `double taiseki;`)
- 文字型 (例 `char letter;`)

●標準入出力の例

- 10 進整数をふたつ、間に空白を置いて読み込む。

  例 `scanf("%d %d", &a, &b);`

- 16 進整数を三つ印字する。

  例 `printf("a + b = %x + %x = %x", a, b, c);`

- 単精度実数型を読み込む。

  例 `scanf("%f", &hankei);`

- 倍精度実数を読み込む。

  例 `scanf("%lf", &taiseki);`

- 実数をすくなくとも 5 文字幅、小数部桁で印字する。

  例 `printf("%5.2f", taiseki);`

- 文字をひとつ読み込む。

  例 `scanf("%c", &letter);`

- 文字を連続して読み込む。

  例 `scanf("%1s", &letter);`

- 文字列を印字する。

  例 `printf("Hello!!!");`

♣ 演習問題 3♣

※ 次節の p 進法の項も参照のこと。

1)GIMP で観たように、モニターの色は RGB それぞれ 0~255 の値をまぜあわせて表現することができる。ホームページで色を指定したいときは、それらを 16 進表示する必要がある。

そこで、RGB それぞれ 0~255 の値を 10 進整数であたえ、それを 16 進で出力するプログラムをつくれ。(気づいたひともいるだろうが、GIMP の色パレットでは 16 進表示もしている。)

2)16 進数での整数の四則演算をおこなえ。すなわち、キーボードから 16 進整数をふたつ読み込んで、それらの和、差、積、商を 16 進整数で出力せよ。また、その出力を 10 進数でも表示せよ。さらに、同様に、8 進数での整数の四則演算を試みよ。

2.3　コンピューター内部での数字と文字のあつかいかた

前節では、各変数の型と入出力をソースファイルでどう記述するかをみた。これは、いわば表面的なことである。実のところ、コンピュータの内部では、これらをどう表現しているのであろうか、という疑問が自然に発生する。この節ではこの問題を調べてみよう。

2.3.1　準備: p-進数 (p-adic numbers) による表示

p-進数 (p-adic numbers) は、Kurt Hensel (1861-1941) によって導入された [16, 17]。C 言語では、とくに、$p = 2, 8, 16$ のときの表記法をしばしば用いる [18, 11]。以下、主として [19] にしたがう。

<< 定義 >>

実数 x を整数 $p > 0$ にかんして

$$x = \pm a_s p^s + a_{s-1}p^{s-1} + \cdots + a_i p^i \cdots + a_0 + \frac{a_{-1}}{p} + \frac{a_{-2}}{p^2} + \cdots + \frac{a_{-j}}{p^j} + \cdots \quad (s = \max\{h : p^h \leq |x|\}) \tag{1}$$

と展開したものを x の p 進展開といい、その係数を

$$\pm a_s a_{s-1} \cdots a_i \cdots a_0 . a_{-1} a_{-2} \cdots a_{-j} \cdots \tag{2}$$

とならべて表わしたものを p-進数 (p-adic numbers) という。x の符号によって $\pm$ をえらぶことは、あきらかである。

このとき、整数部の係数は、

$$a_i = \left\lfloor \frac{x}{p^i} \right\rfloor - p \left\lfloor \frac{x}{p^{i+1}} \right\rfloor \quad (0 \leq i \leq s) \tag{3}$$

となる。また、小数部の係数は、

$$a_j = \lfloor x \cdot p^j - p \lfloor x \cdot p^{j-1} \rfloor \rfloor \quad (j = 1, 2, \cdots) \tag{4}$$

によってあたえられる。ここで、床記号 (floor symbol) $\lfloor y \rfloor$ は、$\lfloor y \rfloor$ = 最大整数 $\leq y$ を意味する。

具体的な計算方法は以下のとおり。(例・参照)

例

$2013.1484375_{10} \rightarrow 7dd.26_{16}$

1) 整数部　　2) 小数部

1) 整数部の係数 a_i の決めかた… x の整数部を p で割った余りを a_0 、その商を p で割った余りを a_1 、さらにその商を p で割った余りを a_2 とする。以下、おなじようにして $a_3, a_4, a_5, \cdots$ を得る。

2) 小数部の係数 a_j の決めかた…x の小数部に p をかけて得られた数の整数部を a_{-1}、その小数部に p をかけて得られた数の整数部を a_{-2} とする。以下、おなじようにして $a_{-3}, a_{-4}, a_{-5}, \cdots$ を得る。

♡ 演習 ♡
1) $\frac{1}{4} = 0.25_{10} \longrightarrow$ 3 進数では？
2) $\frac{1}{10} = 0.1_{10} \longrightarrow$ 2 進数では？
［ 答え: 1) $0.020202\cdots$ 2) $0.0001100110011\cdots$ ］
10 進法で有限桁の数でも、他の基数 (base, radix) で表示すると循環する場合がある。したがって、それを有限桁で打ち切ると誤差が生じる。

2.3.2 情報量とその単位 [9, 20, 11]

bit(ビット) ••• binary digit の略。1 回の ”Yes or No” でわかる情報量。情報量の基本単位。
1 bit=2 進数の 1 桁。1 Byte(バイト) = 8bit = 2 進数の 8 桁 = 16 進数の 2 桁。
1 KB(キロバイト) = 2 の 10 乗バイト。1 MB(メガバイト) = 2 の 20 乗バイト。
1GB(ギガバイト) = 2 の 30 乗バイト。

word(語) ••• プロセッサ (CPU) の命令 (instruction) の長さ (size) を表わす。プロセッサの進化にしたがって、1 word = 8bit → 16bit → 32 bit → 64bit　と変化してきている。(ゲーム機なんか、ずっと前から 128bit である。)
　ネットでソフトをダウンロードするとき、「x86」と表示のあるのは 32 ビット版、「x64」と表示のあるのは 64 ビット版である。

2.3.3 16 進数 [20, 11]

コンピュータのなかでは、文字、数字、画像、音、などすべてが 0 と 1 の列で表わされる。Byte(バイト) のかたまりが、列の読み込み方の基本単位となっている。そこで、その列を 16 進数で表わすと人間には見やすくなる。

10 進数	0	1	2	···	8	9	10	11	12	13	14	15	16	17	18	19	20	···
16 進数	0	1	2	···	8	9	a	b	c	d	e	f	10	11	12	13	14	···

表 3: 16 進数の表現

1 Byte(バイト) は 16 進数 2 桁であらわされる。
例
　1101 1100 (2 進数) = dc (16 進数) = 220 (10 進数)

2.3.4 整数の表現 [11, 21, 22]

int は 4 byte(または、2byte)。
(い) 符号なし整数
非負整数をあらわすときには、たとえば、
```
unsigned int a;
```
とすると、変数 a: $0 \sim 2^{32} - 1 (= 4294967295)$ となる。

(ろ) 符号つき整数
たんに、
`int a;`
とすると、変数 a として、正負、および零の整数をあつかうことができる。
負の数は、2 の補数表示 (two's complement) であらわす。かんたんのために、1 バイト分を模式的に例示してみよう。最高位の桁 (符号ビット) が 0 か 1 かによって、符号をあらわすことになる。

+127	0111 1111
+126	0111 1110
…	…
+1	0000 0001
0	0000 0000
-1	1111 1111
-2	1111 1110
…	…

＊符号ビットが 0 $\longrightarrow$ そのまま 10 進数に変換
＊符号ビットが 1 $\longrightarrow$ 符号はマイナス。すべてのビットを反転してから 1 を加えたものを 10 進数に変換。

2.3.5 実数の表現 [21]

実数は、その近似値であらわす。近似小数をつかうのである。
前の例、$\frac{1}{10} = 0.1_{10} = 0.00011001100110011\cdots$ を 2 進の小数第 5 位までとると、
$0.1_{10} \simeq 0.00011 = 11_2 \cdot 2^{-5} = 3 \cdot 2^{-5} = \frac{3}{32} \simeq 0.09375$、
おなじく、2 進の小数第 10 位までとると、
$0.1_{10} \simeq 0.0001100110 = 1100110_2 \cdot 2^{-10} = 102 \cdot 2^{-10} = \frac{102}{1024} \simeq 0.09961$ となる。
このようにして、有限桁で近似した小数を

$$x \cdot 2^e \qquad (x,\ e \text{ は整数})$$

という形で表わせることがわかる。
実数があたえられたとき、計算機のなかでは、この x(仮数部) 、および、e(指数部) にしかるべきビットの列を収納することになる。この表記では、あたえられた実数の値によって小数点の位置が変化する (浮動小数点表示)。x, e にたいして多くの桁をつかうほど近似の精度がよくなるのは、あきらかであろう。数値計算のときに、float ではなく、double を使うのはこのゆえである。
☆演習☆
x(仮数部)5 バイト、e(指数部) 1 バイトとして、つぎの 10 進数を浮動小数点表示せよ。
1) $\frac{1}{10} = 0.1_{10}$
2) 2013.01_{10}

2.3.6 $p = 16$ 進数での ASCII 文字、および漢字の表示の例

ASCII(ANSI) コード
ASCII=American Standard Code for Information Interchange 。1963 年当時の米国規格協会 (現

在の ANSI=Ameriacan National Standards Institute) が定めた、情報交換用のコード (符号、暗号)。1Byte をつかって、数字と英字 (大文字、小文字)、記号、制御用の特殊文字 (エスケープ・シークエンス) を表わすもの。キーボードのキーに数字を対応させたもの、ともいえる [11]。

コード	···	$0a$	···	20	···	30	31	···	40	41	42	···	61	62	···
キー	···	Enter	···	空白	···	0	1	···	@	A	B	···	a	b	···

表 4: ASCII code の例

先の表でのコードはすべて 16 進表示である。たとえば、キーボードの文字「a」に対応する ASCII コードは、$61 = 0x61 = 61_{16}$ である。

漢字コード

日本語の表記のためには、漢字のコードづけが必要になる。つぎのような幾種類かのコードがつかわれている。

・JIS コード (ISO-2022-JP) ····· 電子メールなどでつかう 7 ビットのコード体系。

・MS 漢字コード (Shift_JIS) ····· パーソナルコンピュータでよくつかわれている 81 ビットのコード体系。

・EUC コード (EUC-JP) = Extended UNIX Code ····· おもに UNIX(あるいはそのパソコン版の Linux 等) でよくつかわれていた 8 ビットのコード体系。いまは、Linux でも下記の UTF-8 をよくつかう。

・Unicode = 世界の主要な文字を一括してあつかうコード。幾種類かあるが、UTF-8 がよくつかわれているようである。Windows Vista も Unicode を取り入れている。

UTF = UCS Transfer Format。UCS = Universal Character Set [23]。具体的には、Windows ならば、『IME パッド』で調べることができる。つぎの図 1 を参照のこと。

※ パソコンでも、OS によって改行コードのちがいがあるが、編集用ソフトやワープロソフト、Web プラウザ (インターネット閲覧ソフト) が自動的に処理してくれる場合もある。「OpenOffice Writer」、「Microsoft Word」、「emacs」、「Firefox」等はこれをおこなってくれる。
(「Internet Explorer」は JIS コードを表示しない可能性があるので注意。)
※ 漢字コードの変換方法については、[8] を参照のこと。
※ 保存するときに、どの OS でも「utf-8」にしておくと漢字変換の**※ Cygwin 端末で漢字変換プログラム「nkf」をつかって駄目押しをしておく**とよい。これをしていると文字化けを気にすることもない。

以上が

漢字表記の進歩 [9]

漢字の表記は、たいせつな問題である。現在普及している漢字コードは、アルファベットをつかう国の文字コードを拡張したものになっている。Unicode は、非西欧圏の文字をも一括してあつかうことを、以前よりも、可能にしてくれている。しかし、それでも、いまのコンピュータの言語のあ

図 2: 漢字コードの例 (『IME パッド』)

つかいかたは、非西欧圏、ことに東アジアの漢字文化圏の言語の表記にはあきらかに適していない。ひとつの漢字の表記が国によって微妙にちがう上に、書体の歴史的な変化もあるからである。この点は、上記の Unicode も解決していない問題である。

コンピュータでの文字の表現 (漢字をふくむ) にかんする Web Page を挙げておく。ここでは、文字の表記をより自然なものにする試みがおこなわれている。いや、さらにすすんで、実用的な製品もでているようである。Web Page をすこし拝んだだけであるが、なかなかりっぱなものである。

1) 京都大学人文科学研究所の「漢字と情報」にかんするページ

●漢字情報研究センター

`http://kanji.zinbun.kyoto-u.ac.jp/`

●出版物

`http://kanji.zinbun.kyoto-u.ac.jp/publications/index.html.ja`

「広報誌『漢字と情報』」。これは、なかなかのものである。

2)『超漢字』トップページ [9, 20]

`http://www.chokanji.com/`

筆者はつかったことはないが、説明を読んだところ、これがいちばん便利なのではないか、という予感がするのである。というのも、初めに述べた日本語表記の問題を完全に解決している模様であるから。たいしたものである。Windows 上でつかえる『超漢字 V』もお目見えしている。つかったことのあるひともいるかもしれない。どうでしたか。

以上、参考になるかもしれない。

参考文献

[1] 授業資料『Cygwin と Linux の手引き (*An Informal Introduction to Cygwin and Linux*』)

[2] [1] の § 3.5。

[3] [1] の § 4.1。

[4] [1] の § 4.2。

[5] B. W. カーニハン & D. M. リッチー (石田 晴久 訳) 『プログラミング言語 C・第 2 版 (The C programming Language)』(共立出版、1989 年)。

[6] [5] の § 1.1。

[7] L. ゴールドシュレーガー & A. リスター (武市・小川・角田　訳)『計算機科学入門・第 2 版 (Computer Science)』、(近代科学社、2000 年) の §2-2。

[8] [1] の § 5.5。

[9] 坂村 健『痛快 コンピュータ学』(集英社文庫、2002 年)。

[10] [5] の § 2.1-§2.4。

[11] J. May & J. Whittle (武舎 広幸 訳) 『Symantec C++ for Macintosh トレーニングブック (Symantec C++ for The Macintosh)』(翔泳社、1994 年) 第 1 章。

[12] D. マーク『Windows ではじめる C プログラミング』(星雲社、2001 年)。

[13] 授業資料『C 言語によるプログラミング 3 (*A Pedestrian Approach to the C Programming Language*)』。

[14] [1] の § 5.3。

[15] [5] の § 1.2。

[16] 『*Abrégé d'histoire des mathématiques*』ed L. Dieudonné (Hermann, Parris, 1978). [17] に引用されている。

[17] R. Rammal, G. Toulouse and M. A. Virasoro, Rev. Mod. Phys. **58**, 765-88 (1986).

[18] [5] の pp 45-6、および pp 234-5。

[19] S. ハルトマン & J. ミクシンスキー（州之内治男・前田功雄 訳）『ルベーグ積分入門』（サイエンス社、1984 年）巻末付録: 訳者補足 2。

[20] 坂村 健『大人のための「情報」教科書』（数研出版、2003 年）。

[21] 和田 秀男『計算数学』（朝倉出版、2000 年）、第 1 章。

[22] 都倉 信樹『新版 情報工学』（放送大学教育振興会、1999 年）、第 3 章。

[23] 西村 めぐみ『図解でわかる Linux のすべて』(日本実業出版社、2000 年)、§ 6.1。