プログラミングおよび実習 I・第 4 回

2003 年 4 月 30 日

1　文字と文字コード

コンピュータの中で文字が扱われるときには、異なる文字に異なる整数を割り当てておき、文字の違いを整数の違いとして扱います。それぞれの文字に割り当てられた整数を、その文字の**文字コード**と呼びます。それぞれの文字に、ある割り当て方に基づいて整数を割り当てることを文字の**符号化**と呼びます。たとえば、 ASCII コードと呼ばれる文字コード (符号化) では、英字や数字等の文字に 0 から 127 までの整数を次の表のように割り当てています。

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
SP	!	"	#	$	%	&	’	(	)	*	+	,	-	.	/
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

(右肩の整数が文字コード)

表 1　ASCII コード

ASCII コードは、コンピュータで文字情報が扱われるときの最も基本的な文字コード (の割り当て方) の規格です。 ASCII コードは、米国で決められた規格ですので、英字や数字、限られた記号などしか表現することができませんが、日本を含め他の地域で使われる文字の文字コードも、多くの場合、この ASCII コードを拡張あるいは若干変更する形で規格が定められています。

ASCII コードでの文字コードが 0 から 31 までと 127 の文字は**制御文字**と呼ばれ、目に見える文字を表すのではなく、通信や表示 (印刷) の書式等を制御するために用いられます。これに対して、英字や数字などの目に見える文字を**図形文字**と呼びます。文字コード 32 の SP はいわゆる「間隔[1] (Space)」を表すもので、画面には何も表示されずにカーソルが 1 文字分進みます。SP は制御文字に分類されることも図形文字に分類されることもあります。

33 から 126 までの文字コード (と図形文字の対応関係) は「ISO 646-US」と呼ばる標準規格となっています。日本では、この「ISO 646-US」の代わりに、「ISO 646-JP」あるいは「JIS X 0201 Roman」と呼ばれる規格の文字コードが用いられることも多くあります。92 と 126 の 2 つの文字コードを除けば、 ISO 646-US (ASCII コード) と ISO 646-JP (JIS X 0201 Roman) との 2 つの規格の間に違いはありません。 ISO 646-JP (JIS X 0201 Roman) では、 92 は「¥」 (円の通貨記号) に、126 は「‾」 (上線) になっています。使用しているコンピュータの環境によって、\ (バックスラッシュ) が ¥ で表示されたり、~ (チルド、波線) が ‾ で表示されたりするのはこのためです。

これらの図形文字の他に、 C によるプログラミングでは、つぎのような制御文字がよく用いられます。

[1] 空白と呼ばれることもよくあります

略号	英名	和名	C での表記[2]	意味や用途
NUL	Null	ナル[3]	\0	C 言語で文字列の終端を表すために使用されます。
BEL	Bell	ベル	\a	印刷装置や表示装置のベルを鳴らすなどして、注意を喚起します。
BS	Backspace	後退	\b	カーソルを (同一行内で)1 文字分戻します。
HT	Horizontal Tabulation	水平タブ	\t	カーソルを同一行内で次の (あらかじめ決められた) タブ位置 (通常 8 文字間隔) まで進めます。
LF	Line feed[4]	改行	\n	C 言語ではカーソルを次の行の行頭へ移動します。一般には、単にカーソルの 1 行分下への移動を表すこともあります。
VT	Vertical Tabulation	垂直タブ	\v	カーソルを次の (あらかじめ決められた) タブ行まで下に進めます。
FF	Form Feed	改ページ[5]	\f	印刷位置を次のページに進めたり、現在の画面を消去してカーソルを初期位置に戻したりします。
CR	Carriage Return	復帰	\r	カーソルを行頭に移動します。

表 2 よく使われる制御文字

2 C での文字処理

2.1 printf による 1 byte 文字の入力

ASCII コードのように、1 byte (= 8 bit) の整数で表現できる文字を関数 printf で出力する場合は、printf の出力書式文字列の中に %c という**変換仕様**を置き、その位置に埋め込みたい文字の文字コード (整数) を、対応する実引数として printf に渡します。たとえば、次のプログラム ascii.c は、32 から 126 までの文字コードと文字の対応関係を 1 行に 8 文字ずつ出力するものです。

ascii.c

```
 1 #include <stdio.h>
 2
 3 int main()
 4 {
 5     int c;
 6
 7     for (c = 32; c <= 126; c++) {
 8         printf("%3d=%c", c, c);
 9         if (c%8 == 7 || c == 126)
10             printf("\n");
11         else
12             printf(" ");
13     }
14     return 0;
15 }
```

プログラム 8 行目の printf の呼び出しでは、%3d と %c のどちらの変換仕様も、変数 c の値に基づいた出力をさせていますが、%3d では c の値の十進数表記 (いくつかの数字の並び) が、%c では c の値を

[2] 4 – 3ページの 2.2 節「文字定数」を参照してください

[3]「空白」と呼ばれることもありますが、SP (スペース) と混同しやすいので「ナル」と呼びましょう

[4]「New Line」とも呼ばれます

[5]「書式送り」とも呼ばれます

文字コードとする 1 つの文字が出力されることに注意してください。このプログラムをコンパイルして実行すると次のようになります。

ascii.c の実行例

```
s1609h017% cc ascii.c -o ascii
s1609h017% ./ascii
 32=   33=!  34="  35=#  36=$  37=%  38=&  39='
 40=(  41=)  42=*  43=+  44=,  45=-  46=.  47=/
 48=0  49=1  50=2  51=3  52=4  53=5  54=6  55=7
 56=8  57=9  58=:  59=;  60=<  61==  62=>  63=?
 64=@  65=A  66=B  67=C  68=D  69=E  70=F  71=G
 72=H  73=I  74=J  75=K  76=L  77=M  78=N  79=O
 80=P  81=Q  82=R  83=S  84=T  85=U  86=V  87=W
 88=X  89=Y  90=Z  91=[  92=\  93=]  94=^  95=_
 96=`  97=a  98=b  99=c 100=d 101=e 102=f 103=g
104=h 105=i 106=j 107=k 108=l 109=m 110=n 111=o
112=p 113=q 114=r 115=s 116=t 117=u 118=v 119=w
120=x 121=y 122=z 123={ 124=| 125=} 126=~
s1609h017%
```

2.2 文字定数

C のプログラム中で、特定の文字の文字コードを利用したい場合、その文字を「'」(一重引用符) で囲むことによって、囲まれた文字の文字コード (int 型の整数値) を表すことができます。たとえば、C のプログラム中で 'A' と書くと、英文字 A の文字コード (ASCII コードが使用されている場合は 65) を表します。このような表記によるプログラム中の定数を**文字定数**と呼びます。このとき、4 – 2ページの表 2 にあるような制御文字は、表中の「C での表記」を「'」で囲み、たとえば、改行文字の文字コードは '\n' のように書きます。「'」という文字 (一重引用符) 自体の文字コードは '\'' で、「\」(バックスラッシュ) の文字コードは '\\' で表すことができます。

ASCII コードを使っている場合、プログラム ascii.c は次のように書いても同じように動きます。

ascii2.c

```
 1 #include <stdio.h>
 2
 3 int main()
 4 {
 5     int c;
 6
 7     for (c = ' '; c <= '~'; c++)
 8         printf("%3d=%c%c", c, c, (c%8 == 7 || c == '~') ? '\n' : ' ');
 9     return 0;
10 }
```

2.3 char 型

ASCII コードで符号化された文字は 0 から 127 の整数で表現されますが、C には、この範囲の整数を扱うのに都合のよい char 型というデータ型が用意されています。char 型のデータは 1 byte (= 8

bit) で表現できる符号付きあるいは符号なし[6]の整数です。符号付きの場合は −128 から 127 まで[7]、符号なしの場合は 0 から 255 までの範囲の整数を表現することができ、整数に関する四則演算等を char 型のデータに対しても行うことができます。ただし、式の中に現れた char 型のデータは、まず int 型に変換されてから計算が行われます。

2.4 char 型の変数

char 型のデータを格納する変数は、int 型の変数を宣言するのと同様に、C の予約語 char を使ってつぎのように宣言します。

```
char ch;
```

1 つの char 型の変数は、1 つの char 型のデータ (整数) を記憶することができます。記憶できる数値の範囲が狭いことを除けば、int 型の変数と全く同じように代入したり参照したりすることが可能です。

たとえば、先のプログラム ascii.c や ascii2.c の 5 行目の「int c;」は「char c;」としても全く同じように動きます。これは、変数 c で扱う範囲の整数 (文字コード) が char 型で十分表現可能であり、また、関数 printf の第 2 引数以降に char 型のデータが現れた場合、ちょうど式の計算の中で char 型のデータが int 型へ変換されてから計算が行われるのと同じように、まず int 型に変換されてから関数 printf へ渡されるからです。

2.5 scanf による 1 byte 文字の入力

関数 scanf を使って、1 byte の整数で表現された文字を入力することもできます。入力書式文字列の中には %c という変換仕様を書きます。ちょうどその変換仕様の位置に現れた (入力された) 文字の文字コードが、対応する (scanf の) 実引数で指定された変数等に格納されます。変換仕様 %c に対応する実引数は char 型へのポインタ (char 型のデータの記憶領域のアドレス) でなければなりません。

次は、入力された行中の文字の文字コードをすべて表示するプログラム chars.c とその実行例です。プログラムの 9 行目では、関数 scanf に char 型のデータの格納場所 (アドレス) を渡さなければなりませんので、5 行目の「char c;」を「int c;」とすることはできません。

chars.c

```
 1 #include <stdio.h>
 2
 3 int main()
 4 {
 5     char c;
 6
 7     printf("文字列を入力してください = ");
 8     do {
 9         scanf("%c", &c);
10         printf("%3d=%c\n", c, c);
11     } while (c != '\n');
12     return 0;
13 }
```

[6]符号付きか符号なしかは、C の処理系によって違っています

[7]C の規格上は −127 から 127 までとなっている可能性もあります

chars.c の実行例

```
s1609h017% ./chars
文字列を入力してください = This is a test.
 84=T
104=h
105=i
115=s
 32=
105=i
115=s
 32=
 97=a
 32=
116=t
101=e
115=s
116=t
 46=.
 10=

s1609h017%
```

プログラム chars.c の実行例で、キーボードからの入力がすべて完了してからプログラムの出力が始まっているのは、改行 (Enter) キーが押されるまでは、キーボードから入力された文字列は実行中のプログラムに渡されないからです。chars の実行が開始されると、7 行目の printf の呼び出しで「文字列を入力してください =」という出力を行った後、改行 (Enter) キーが押されるまでは、9 行目で始めて関数 scanf が呼び出された状態で停止しています。ユーザーが「This is a test.」に続いて、改行 (Enter) キーを押したときに始めて、T から改行文字までの 16 文字をプログラムが読み取ることが可能となり、scanf は先頭の文字 T の文字コードを変数 c に格納して、関数 main に復帰します。それ以降の scanf の呼び出しはプログラムを停止させる事なく直ちに完了し、入力された文字が順に処理されていきます。

関数 scanf の変換仕様として、%d、%f、%lf など、数値を読み取るものを使用すると、 入力された数値に先行する空白類 (スペース、水平タブ、改行等) は読み飛ばされ (無視され) ますが、%c の場合は空白類も 1 つの文字として、その文字コードが変数等に格納されることに注意してください。

空白類を無視して文字を入力したい場合は、%c の前に空白を置いて、入力書式文字列を " %c" のようにします。入力書式文字列に空白が現れている場合、scanfは、その位置に現れた (入力された) 空白類をすべて読み飛ばします。

3 オセロゲームの列番号を英文字にする

次のプログラム othello09.c は、前回の演習問題で作成した othello08.c (4 – 11 ページの付録を参照) を変更して、1 から 8 までの列番号を a から h までの英文字で表すように変更したものです。* 印を付けた行が変更されています。

othello09.c

```
 1 #include <stdio.h>
 2
 3 #define BLACK           (1)         /* 黒石を表す定数 */
 4 #define WHITE           (-BLACK)    /* 白石を表す定数 */
 5 #define NONE            (0)         /* 石がないことを表す */
```

```
 6 #define OPPONENT(t)     (-(t))      /* t の相手 */
 7 #define BOARD_SIZE      (8)         /* オセロボードの縦横のマス数 */
 8
 9 /* 盤面を初期化する */
10 void reset(int b[][BOARD_SIZE])
11 {
12     int x, y;
13
14     /* すべてのマスをクリアする */
15     for (x = 0; x < BOARD_SIZE; x++)
16         for (y = 0; y < BOARD_SIZE; y++)
17             b[x][y] = NONE;
18     /* 中央に4つの石を置く */
19     b[BOARD_SIZE/2-1][BOARD_SIZE/2-1] = WHITE;
20     b[BOARD_SIZE/2-1][BOARD_SIZE/2] = BLACK;
21     b[BOARD_SIZE/2][BOARD_SIZE/2-1] = BLACK;
22     b[BOARD_SIZE/2][BOARD_SIZE/2] = WHITE;
23 }
24
25 /* 盤面を表示する */
26 void showBoard(int b[][BOARD_SIZE])
27 {
28     int x, y;
29
30     /* 列番号を表示する */
31     for (x = 0; x < BOARD_SIZE; x++)
*32         printf(" %c", x + 'a');
33     printf("\n");
34     /* 各行を表示する */
35     for (y = 0; y < BOARD_SIZE; y++) {
*36         printf("%c", y + '1');
37         for (x = 0; x < BOARD_SIZE; x++) {
38             if (b[x][y] == BLACK)
39                 printf("●");
40             else if (b[x][y] == WHITE)
41                 printf("○");
42             else
43                 printf("□");
44         }
45         printf("\n");
46     }
47 }
48
49 /* ユーザーが指定した位置に石を置く */
50 void inputMove(int b[][BOARD_SIZE], int turn)
51 {
*52     char ch;
53     int x, y;
54
55     while (1) {
56         /* 列番号を入力する */
57         do {
*58             ch = ' ';
59             printf("列 = ");
*60             scanf(" %c", &ch);
*61             x = ch - 'a';
*62         } while (x < 0 || BOARD_SIZE <= x);
63         /* 行番号を入力する */
64         do {
*65             ch = ' ';
66             printf("行 = ");
*67             scanf(" %c", &ch);
*68             y = ch - '1';
```

```
*69         } while (y < 0 || BOARD_SIZE <= y);
*70         if (b[x][y] == NONE)
 71             break;
 72         printf("そこには置けません\n");
 73     }
 74     /* 石を置く */
*75     b[x][y] = turn;
 76 }
 77
 78 int main()
 79 {
 80     /* 盤面の状態 (BLACK/WHITE/NONE) */
 81     int board[BOARD_SIZE][BOARD_SIZE];
 82     /* つぎの手番 */
 83     int turn = BLACK;
 84
 85     reset(board);
 86     while (1) {
 87         showBoard(board);
 88         inputMove(board, turn);
 89         turn = OPPONENT(turn);
 90     }
 91     /* プログラムを終了する */
 92     return 0;
 93 }
```

このプログラムは、ASCII コードのように、a から h までの文字コードと、1 から 8 までの文字コードが連続していることを前提として書かれています。このプログラムをコンパイルして実行すると次のようになります。

othello09 の実行例

```
s1609h017% ./othello09
 a b c d e f g h
1□□□□□□□□
2□□□□□□□□
3□□□□□□□□
4□□□○●□□□
5□□□●○□□□
6□□□□□□□□
7□□□□□□□□
8□□□□□□□□
列 = c
行 = 2
 a b c d e f g h
1□□□□□□□□
2□□●□□□□□
3□□□□□□□□
4□□□○●□□□
5□□□●○□□□
6□□□□□□□□
7□□□□□□□□
8□□□□□□□□
列 = x
列 = g
行 = 4
 a b c d e f g h
1□□□□□□□□
2□□●□□□□□
3□□□□□□□□
```

```
4□□□○●□○□
5□□□●○□□□
6□□□□□□□□
7□□□□□□□□
8□□□□□□□□
列 = d
行 = 0
行 = 5
そこには置けません
列 = d
行 = 7
 a b c d e f g h
1□□□□□□□□
2□□●□□□□□
3□□□□□□□□
4□□□○●□○□
5□□□●○□□□
6□□□□□□□□
7□□□●□□□□
8□□□□□□□□
列 = ^C
s1609h017%
```

4 演習問題

4.1 例題プログラムを試す

講義で解説した othello09.c というプログラムを自分で作り[8]、コンパイル、実行してみなさい。完成したら、「mprog1 othello09.c」のようにして mprog1 コマンドを実行し、正しくできているかどうかチェックしてください。正しくできている場合は、そのプログラムが提出されたことになります。プログラムは Prog1 というディレクトリに作ることを忘れないようにしましょう。以下の 2 つの演習問題も同様です。

4.2 1 行の入力で位置が指定できるようにする

まず、othello09.c を othello10.c にコピーしなさい。その後 othello10.c の関数 inputMove の定義を変更して、次の実行例のように、列と行の指定を英小文字と数字の 2 文字で 1 度に行うようにしなさい。ただし、列や行の指定が a〜h や 1〜8 の範囲に収まっていなかった場合や、指定された位置にすでに石がある場合は、どちらも「そこには置けません」というメッセージを表示して、ユーザーにもう一度入力を促すようにしなさい。

「d4」のように入力された 2 つの文字 (の文字コード) を、たとえば char 型の 2 つの変数 ch1 と ch2 にそれぞれ格納するには、次のように scanf を呼び出します[9]。

```
scanf(" %c %c", &ch1, &ch2);
```

scanf の入力書式文字列の 2 つの %c の前に空白文字があるのは、空白やタブ文字、改行文字などを無視して文字を読み込みたいからです。

---

[8] 前回作成した othello08.c をコピーして変更すると簡単かも知れません

[9]「scanf(" %c", &ch1);」に続いて「scanf(" %c", &ch2);」を実行しても同じです

othello10 の 実行例

```
s1609h017% ./othello10
 a b c d e f g h
1□□□□□□□□
2□□□□□□□□
3□□□□□□□□
4□□□○●□□□
5□□□●○□□□
6□□□□□□□□
7□□□□□□□□
8□□□□□□□□
位置 = d4
そこには置けません
位置 = d0
そこには置けません
位置 = d1
 a b c d e f g h
1□□□●□□□□
2□□□□□□□□
3□□□□□□□□
4□□□○●□□□
5□□□●○□□□
6□□□□□□□□
7□□□□□□□□
8□□□□□□□□
位置 = x7
そこには置けません
位置 = g7
 a b c d e f g h
1□□□●□□□□
2□□□□□□□□
3□□□□□□□□
4□□□○●□□□
5□□□●○□□□
6□□□□□□□□
7□□□□□□○□
8□□□□□□□□
位置 = ^C
s1609h017%
```

4.3 列と行の 2 文字の順番を交換できるようにする

まず、othello10.c を othello11.c にコピーしなさい。その後 othello11.c の関数 inputMove の定義を変更して、次の実行例のように、「列行」の順の入力でも、「行列」の順の入力でも、石を置くマスが指定できるようにしなさい。

othello11 の実行例

```
s1609h017% ./othello11
 a b c d e f g h
1□□□□□□□□
2□□□□□□□□
3□□□□□□□□
4□□□○●□□□
5□□□●○□□□
6□□□□□□□□
7□□□□□□□□
```

```
8□□□□□□□□
位置 = 2d
 a b c d e f g h
1□□□□□□□□
2□□□●□□□□
3□□□□□□□□
4□□□○●□□□
5□□□●○□□□
6□□□□□□□□
7□□□□□□□□
8□□□□□□□□
位置 = g5
 a b c d e f g h
1□□□□□□□□
2□□□●□□□□
3□□□□□□□□
4□□□○●□□□
5□□□●○□○□
6□□□□□□□□
7□□□□□□□□
8□□□□□□□□
位置 = 9e
そこには置けません
位置 = 8e
 a b c d e f g h
1□□□□□□□□
2□□□●□□□□
3□□□□□□□□
4□□□○●□□□
5□□□●○□○□
6□□□□□□□□
7□□□□□□□□
8□□□□●□□□
位置 = ^C
s1609h017%
```

次回の予定

次回は `othello11.c` をさらに発展させて、つぎのようなことを行う予定です。

- `getchar` という標準入出力ライブラリ関数で、キーボードからの文字を 1 文字ずつ読み取って処理する方法を勉強します。
- 文字の配列として文字列を扱う方法を勉強します。
- `q` という文字を入力すると、プログラムが終了するようにします。

プログラミングおよび実習 I・第 4 回・終り

付録: 前回の演習問題 othello08.c のプログラム例

othello08.c

```
 1 #include <stdio.h>
 2
 3 #define BLACK           (1)         /* 黒石を表す定数 */
 4 #define WHITE           (-BLACK)    /* 白石を表す定数 */
 5 #define NONE            (0)         /* 石がないことを表す */
 6 #define OPPONENT(t)     (-(t))      /* t の相手 */
 7 #define BOARD_SIZE      (8)         /* オセロボードの縦横のマス数 */
 8
 9 /* 盤面を初期化する */
10 void reset(int b[][BOARD_SIZE])
11 {
12     int x, y;
13
14     /* すべてのマスをクリアする */
15     for (x = 0; x < BOARD_SIZE; x++)
16         for (y = 0; y < BOARD_SIZE; y++)
17             b[x][y] = NONE;
18     /* 中央に4つの石を置く */
19     b[BOARD_SIZE/2-1][BOARD_SIZE/2-1] = WHITE;
20     b[BOARD_SIZE/2-1][BOARD_SIZE/2] = BLACK;
21     b[BOARD_SIZE/2][BOARD_SIZE/2-1] = BLACK;
22     b[BOARD_SIZE/2][BOARD_SIZE/2] = WHITE;
23 }
24
25 /* 盤面を表示する */
26 void showBoard(int b[][BOARD_SIZE])
27 {
28     int x, y;
29
30     /* 列番号を表示する */
31     for (x = 0; x < BOARD_SIZE; x++)
32         printf(" %d", x+1);
33     printf("\n");
34     /* 各行を表示する */
35     for (y = 0; y < BOARD_SIZE; y++) {
36         printf("%d", y+1);
37         for (x = 0; x < BOARD_SIZE; x++) {
38             if (b[x][y] == BLACK)
39                 printf("●");
40             else if (b[x][y] == WHITE)
41                 printf("○");
42             else
43                 printf("□");
44         }
45         printf("\n");
46     }
47 }
48
49 /* ユーザーが指定した位置に石を置く */
50 void inputMove(int b[][BOARD_SIZE], int turn)
51 {
52     int x, y;
53
54     while (1) {
55         /* 列番号を入力する */
56         do {
```

```
57              x = 0;
58              printf("列 = ");
59              scanf("%d", &x);
60          } while (x <= 0 || BOARD_SIZE < x);
61          /* 行番号を入力する */
62          do {
63              y = 0;
64              printf("行 = ");
65              scanf("%d", &y);
66          } while (y <= 0 || BOARD_SIZE < y);
67          if (b[x-1][y-1] == NONE)
68              break;
69          printf("そこには置けません\n");
70      }
71      /* 石を置く */
72      b[x-1][y-1] = turn;
73  }
74
75  int main()
76  {
77      /* 盤面の状態 (BLACK/WHITE/NONE) */
78      int board[BOARD_SIZE][BOARD_SIZE];
79      /* つぎの手番 */
80      int turn = BLACK;
81
82      reset(board);
83      while (1) {
84          showBoard(board);
85          inputMove(board, turn);
86          turn = OPPONENT(turn);
87      }
88      /* プログラムを終了する */
89      return 0;
90  }
```