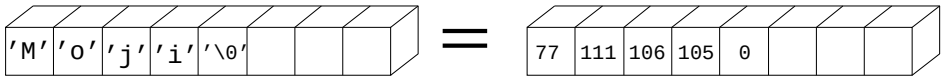
システム科学科 情報処理演習テキスト 8
– 文字列 –

原案:佐藤宏介 **修正**: 升谷 保博, 庄野 逸, 鳩野逸生, 橋本守, 才脇直樹，田中宏喜，日浦慎作

ポイント 1: 文字列は、文字コードを要素とし、'\0' で終端される 1 次元配列として取り扱われる。宣言の際には配列の要素数は文字数+1 が必要であることに注意

宣言: `char 文字列名 [文字数+1];` 例: `char moji[10];` `char string[100];`

```
char str[8] = "Moji";
```



ポイント 2: 文字列を表す配列は、上図のように初期化できる。通常の配列の初期化と同様に、要素数 (文字数+1) は省略しても良い

```
/* 右のプログラムとほぼ同じ */
#include <stdio.h>

int main( void )
{
    char moji[] = "Moji1";
    printf( "String = %s\n", moji );
    return 0;
}
```

```
/* 左のプログラムとほぼ同じ */
#include <stdio.h>

int main( void )
{
    char moji[6];

    moji[0] = 'M';
    moji[1] = 'o';
    moji[2] = 'j';
    moji[3] = 'i';
    moji[4] = '1';     /* =1 ではないことに注意 */
    moji[5] = '\0';    /* 終端文字 */
    printf( "String = %s\n", moji );
    return 0;
}
```

ポイント 3: `printf()` 関数の書式指定文字列の中の "`%s`" は、対応する引数に指定された文字列を出力する。

ポイント 4: 文字コードは、日本語モードにせずに入力可能な英数字、記号に 1 対 1 で対応する 1 byte の数値である。文字コードは、ASCII コード表が使用されることが多い。日本語は 2bytes で 1 文字を表現する。

ポイント 5: 英数字一文字を "'" (アポストロフィ、もしくはシングルクォート) で囲んだものは、その文字の文字コードを値とする定数である

`'0'` → 48　`'a'` → 97　`'('` → 40
`'1'` → 49　`'b'` → 98　`')'` → 41

ASCII コード表

下位 4 ビット	上位 4 ビット							
	0	1	2	3	4	5	6	7
$0_{(16)}$	NUL	DEL	SP	0	@	P	‘	p
$1_{(16)}$	SOH	DC1	!	1	A	Q	a	q
$2_{(16)}$	STX	DC2	”	2	B	R	b	r
$3_{(16)}$	ETX	DC3	#	3	C	S	c	s
$4_{(16)}$	EOT	DC4	$	4	D	T	d	t
$5_{(16)}$	ENQ	NAC	%	5	E	U	e	u
$6_{(16)}$	ACK	SYN	&	6	F	V	f	v
$7_{(16)}$	BEL	ETB	’	7	G	W	g	w
$8_{(16)}$	BS	CAN	(	8	H	X	h	x
$9_{(16)}$	HT	EM	)	9	I	Y	i	y
$A_{(16)}$	LF	SUB	*	:	J	Z	j	z
$B_{(16)}$	VT	ESC	+	;	K	[	k	{
$C_{(16)}$	FF	FS	,	<	L	\	l	|
$D_{(16)}$	CR	GS	-	=	M	]	m	}
$E_{(16)}$	SO	RS	.	>	N	^	n	~
$F_{(16)}$	S1	US	/	?	O	_	o	DEL

ポイント 6: 画面上に表示できない文字 (制御文字) は、“\英数字” で表す。’\0’ (終端文字)、’\n’ (改行文字) は、それぞれ 0 と 10 という数値に対応する。

\ a	アラート (ベル:BEL)
\ b	1 文字後退 (バック・スペース:BS)
\ f	改頁 (フォーム・フィード:FF)
\ n	復帰改行 (ニューライン:NL)
\ r	復帰 (キャリッジ・リターン:CR)
\ t	水平タブ (ホリゾンタル・タブレーション:HT)
\ v	垂直タブ (バーティカル・タブレーション:VT)
\ \	逆スラッシュ
\ ’	一重引用符 (シングル・クォーテーション:’)
\ ”	二重引用符 (ダブル・クォーテーション:”)
\ 0	ヌル・コード (NULL)

```
/* 文字と文字コードを出力 */
#include <stdio.h>

int main( void )
{
    char suji[] = "123";
    char moji[] = "ABC\n";
    int i;

    for( i = 0 ; suji[i] != '\0' ; i++ )
        printf( "%c -> %d (%x)\n", suji[i], (int)suji[i], (int)suji[i] );
    for( i = 0 ; moji[i] != '\0' ; i++ )
        printf( "%c -> %d (%x)\n", moji[i], (int)moji[i], (int)moji[i] );

    return 0;
}
```

左プログラムの出力結果:

```
1 -> 49 (31)
2 -> 50 (32)
3 -> 51 (33)
A -> 65 (41)
B -> 66 (42)
C -> 67 (43)

 -> 10 (a)
```

ポイント 7: printf() 関数の書式指定文字列の中の “%c” は、対応する引数に指定された文字コードの文字を 1 文字出力する。“%x” は、対応する引数に指定された整数を 16 進数で表示する。

ポイント 8: キーボードから 1 文字入力するためには、getchar() 関数を用いる。getchar() 関数の戻り値は文字コードである。getchar() 関数が入力終了文字 (Control + D) を検知した場合, EOF(通常は -1) を値として返す。
(HandyMake 上で 終了文字を 入力するためには、Control + Q とした後に Control + D と入力します。これは、HandyMake 特有の事情です。)

ポイント 9: EOF は、文字コードの範囲 (0 ∼ 255) に含まれない数値のため, getchar() 関数の戻り値は、**文字 (char) 型** ではなく**整数 (int) 型**である。

example8_1.c

```
/*
 * 読み込んだ内容を文字ごとに表示
 */
#include <stdio.h>

int main(void)
{
    int c;                          /* char ではいけない! */

    printf("文字列を入力して最後に Return キーを押して下さい. \n");
    printf("(終了は Control-d)\n");

    /* キーボード入力は Return を押した時に 1 度にまとめて渡される */
    while ( (c = getchar()) != EOF ) {
        printf("'%c' -> %3d (%xh)\n", c, (int)c, (int)c);
    }
    return 0;
}
```

ポイント 10: データの型を明示的に行う場合には “cast 演算子” を用いる。cast の一般形は “**(型名) 式**” である。

例
```
int a; double b; char c;   /* 変数の宣言 */
b = (double)a;             /* 実数型に変換 */
a = (int)b;                /* 整数型に変換 */
d = (char)a;               /* 文字型に変換 */
```

ポイント 11: 文字配列を関数としてデータとして渡すには、引数に文字配列の名前を書けば良い (通常の配列と同じ扱い)

ポイント 12: 文字配列を関数にデータとして渡した場合、関数中で対応文字配列の要素を書き換えれば、呼び出し側の文字配列データも書き換わる。(通常の配列と同じ扱い)

example8_2.c

```
#include <stdio.h>

int mygetline(char s[], int size);
int main(void)
{
    char sei[20], mei[20];

    printf("姓を入力して下さい → "); mygetline(sei, 20);
    printf("名を入力して下さい → "); mygetline(mei, 20);

    printf("あなたは %s %s さんですね. \n", sei, mei);

    return 0;
}

/*
 * 改行までキーボード入力を読み込んで改行直前までを文字列 s に格納する.
 * 改行がなくても, size-1 文字読み込んだら, そこまでを文字列に格納し, 処理を終了する.
 * 戻り値は, 読み込んだ文字数をとする.
 * 入力終了 (EOF) を検知して文字を 1 文字も読み込めなかった場合は, -1 を戻り値とする.
 */
int mygetline(char s[], int size)
{
    int         c, i;

    for ( i = 0;  (c = getchar()) != EOF && c != '\n' && i < size - 1; i++ ) {
        s[i] = c;
    }

    s[i] = '\0';

    if ( c == EOF && i == 0 ){
        return -1;
    }

    return i;
}
```

ポイント 13: C 言語の標準ライブラリには文字や文字列処理の関数が定義されている。

<string.h> に含まれる代表的な文字列処理関数

strcat	文字列の連結
strcmp	文字列の比較
strcpy	文字列のコピー
strlen	文字列の長さを調べる
strcat	文字列の連結 (文字列の長さ制限あり)
strcmp	文字列の比較 (文字列の長さ制限あり)
strcpy	文字列のコピー (文字列の長さ制限あり)

<ctype.h> に含まれる代表的な文字処理関数

isalnum	英字または数字であるかを調べる
isalpha	アルファベットかどうか調べる
isspace	空白文字であるかを調べる
isdigit	数字 (0 から 9 まで) かどうかを調べる
islower	小文字かどうかを調べる
isupper	大文字かどうかを調べる
toupper	大文字に変換する
tolower	小文字に変換する

コンソール画面で、“man 関数名” をタイプして各関数の詳細について調べること

example8_3.c

```
/*
 * 文字列の中の小文字を大文字に変換する関数の実験
 */
#include <stdio.h>

int mygetline(char s[], int size);
int oomoji(char s[]);

int main(void)
{
    char        str[100];
    int         ret;

    while ( 1 ) {
        printf("\n 入力 → ");
        if ( mygetline(str, 100) == -1 ) {
            break;
        }
        ret = oomoji(str);
        printf("出力   → %s\n", str);
        printf("戻り値 → %d\n", ret);
    }
    printf("\n お疲れ様でした. \n");

    return 0;
}

/*
 * example82.c などを参照
 */
int mygetline(char s[], int size)
{
    int         c, i;

    for ( i = 0;  (c = getchar()) != EOF && c != '\n' && i < size - 1; i++ ) {
        s[i] = c;
    }

    s[i] = '\0';

    if ( c == EOF && i == 0 ){
        return -1;
    }

    return i;
}

/*
 * 文字列 s の中の'a'〜'z' の文字を'A'〜'Z' に置き換える.
 * 戻り値は置き換えた文字数.
 */
int oomoji(char s[])
{
    int         i, r = 0;

    for ( i = 0; s[i] != '\0'; i++ ) {
        if ( s[i] >= 'a' && s[i] <= 'z' ) {
            s[i] += 'A' - 'a';
            r++;
        }
    }

    return r;
}
```