第２章

表示に凝ろう

本章で作るのは、《テロップ》や《暗算トレーニング》などのプログラムです。時間をあやつったり、文字を消したり動かしたりするプログラムの作成を通じて、時間や表示に関する各種のテクニックを身につけます。

この章で学ぶおもなこと

- 拡張表記
- 警報
- 改行と復帰
- 後退
- タブ
- 引用符
- 表示済み文字の消去と書きかえ
- 一定時間の処理停止
- 処理時間の計測
- キャスト
- 文字列
- ナル文字
- typedef宣言
- 書式付き入出力
- ⊙clock_t 型
- ⊙clock 関数
- ⊙printf 関数
- ⊙putchar 関数
- ⊙scanf 関数
- ⊙strlen 関数
- ⊙CLOCKS_PER_SEC

2-1 拡張表記を使いこなす

拡張表記を活用すれば、画面上の文字を消したり動かしたりと、さまざまな表示効果を生み出せます。本節では、拡張表記のマスターを目指して学習します。

拡張表記

拡張表記（*escape sequence*）は、逆斜線記号＼を先頭にした文字の並びによって、単一の文字を表す表記法です。その一覧を **Table 2-1** に示します。

まずは、これらを使いこなせるように学習を進めていきましょう。

Table 2-1 拡張表記（escape sequence）

■ **単純拡張表記**（*simple escape sequence*）		
`\a`	**警報**（*alert*）	聴覚的または視覚的な警報を発する。
`\b`	**後退**（*backspace*）	直前の位置へ移動する。
`\f`	**書式送り**（*form feed*）	改ページして、次のページの先頭へ移動する。
`\n`	**改行**（*new line*）	改行して、次の行の先頭へ移動する。
`\r`	**復帰**（*carriage return*）	現在の行の先頭位置へ移動する。
`\t`	**水平タブ**（*horizontal tab*）	次の水平タブ位置へ移動する。
`\v`	**垂直タブ**（*vertical tab*）	次の垂直タブ位置へ移動する。
`\\`	**文字** \	
`\?`	**文字** ?	
`\'`	**文字** '	
`\"`	**文字** "	
■ **8進拡張表記**（*octal escape sequence*）		
`\ooo`	*ooo* は1～3桁の8進数	8進数で *ooo* の値をもつ文字。
■ **16進拡張表記**（*hexadecimal escape sequence*）		
`\xhh`	*hh* は任意の桁数の16進数	16進数で *hh* の値をもつ文字。

▶ 拡張表記は2個以上の文字で構成されますが、それによって表されるのは、あくまでも単一の文字です。なお、多くのパソコンで利用されているJISコードでは、逆斜線＼の代わりに円記号 ¥ を利用しなければなりません。

\a … 警報

警報 `\a` を出力すると、**“聴覚的または視覚的な警報”**が発せられます。実際には、多くの環境で“ビープ音”が鳴ります（音を出さずに画面を点滅させる環境もあります）。

なお、警報を出力しても**現表示位置**（コンソール画面における**カーソル**の位置）が変更されることはありません。

▶ 本書では、警報の出力結果を♪と表すのでしたね（p.2）。

\n … 改行

改行 \n を出力すると、現表示位置が“**次の行の先頭**”に移動します。

警報と改行を出力するプログラム例を **List 2-1** に示します。

List 2-1 chap02/alertnl.c

```
/* 警報\aと改行\nを出力 */

#include <stdio.h>

int main(void)
{
    printf("こんにちは。\n初めまして。\n");
    printf("\a警告します。\n\n");
    printf("\a\a今度は２回警告します。\n");

    return (0);
}
```

途中で改行

警報を発する

１行空ける

実行結果

```
こんにちは。
初めまして。
♪警告します。

♪♪今度は２回警告します。
```

『こんにちは。』に続いて改行文字 \n を出力するため、**Fig.2-1** に示すように『初めまして。』は次の行に表示されます。

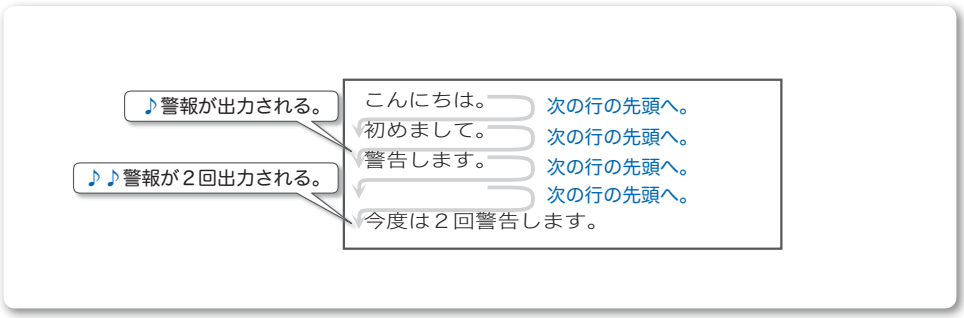


Fig.2-1 警報\aと改行\nの働き

２番目に表示する文字列の末尾には、２個の連続した改行文字があります。そのため、最後に出力する『今度は２回警告します。』は、１行空いたところに表示されます。

▶ 連続した改行文字によって空の行を出力するテクニックは、第１章の《数当てゲーム》でも利用していました。

\f … 書式送り

書式送りを出力すると、現表示位置が“**次の論理ページの先頭位置**”に移動します。通常の環境では、書式送りをコンソール画面へ出力しても何も起こりません。

プリンタへの出力において、改ページを行う際に利用します。

\b … 後退

後退 \b を出力すると、現表示位置が“**その行内での直前の位置**”に移動します。

▶ 現表示位置が行の先頭であるときに、後退を出力するとどうなるかは規定されていません。というのも、前の行（上の行）にカーソルを戻せない環境があるからです。

後退によって**動きのある表示効果**を生み出す例が **List 2-2** です。まずは実行してみましょう。最初に文字列 "ABCDEFG" が表示されます。その後、1 秒ごとに末尾側から 1 文字ずつ消えていき、全部の文字が消えてしまったらプログラムが終了します。

List 2-2 chap02/backspace.c

```
/* 後退\bの利用例：１秒ごとに１文字ずつ消去 */

#include <time.h>
#include <stdio.h>

/*--- xミリ秒経過するのを待つ ---*/
int sleep(unsigned long x)
{
    clock_t c1 = clock(), c2;

    do {
        if ((c2 = clock()) == (clock_t)-1)  /* エラー */
            return (0);
    } while (1000.0 * (c2 - c1) / CLOCKS_PER_SEC < x);
    return (1);
}

int main(void)
{
    int i;

    printf("ABCDEFG");

    for (i = 0; i < 7; i++) {
        sleep(1000);                /* 後ろから１文字ずつ */
        printf("\b \b");            /* １秒ごとに消す     */
        fflush(stdout);             /* バッファを掃き出す */
    }

    return (0);
}
```

xミリ秒だけ時間をつぶす関数

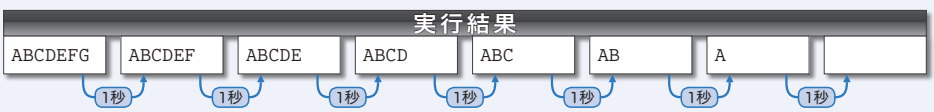
1

2

3

出力を確実に行う

実行結果



1 の関数 *sleep* の働きは、***x* ミリ秒だけ時間をつぶす**（処理を停止する）ことです。2 のように *sleep*(1000) と呼び出されると、約 1 秒経過してから呼出し元に戻ります。

この関数の詳細は p.48 で学習します。それまでは、

『*sleep*(*x*) と呼び出せば、*x* ミリ秒だけ時間をつぶせる。』

と覚えておくとよいでしょう。

main 関数を理解していきましょう。まず **"ABCDEFG"** を表示します。その後、**for** 文の働きによって、**"\b \b"** を 1 秒間隔で 7 回出力しています。このことから、**"\b \b"** を出力すると、**表示済み文字の末尾文字が消える**ことが分かります。文字を消す原理を示したのが **Fig.2-2** です。

▶ この図が示すのは、**for** 文の繰返しの第 1 回目における、末尾文字 **'G'** の消去の様子です。

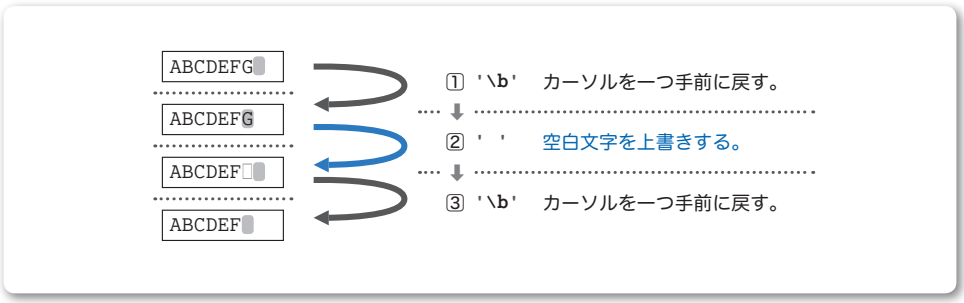


Fig.2-2 後退\bによる文字の消去

文字列 **"ABCDEFG"** を表示した時点で、カーソルは文字 **'G'** の直後に位置しています。この状態から、以下の手順で文字 **'G'** を消去します。

1. 後退 **'\b'** を出力：カーソルを一つ戻して **'G'** の位置へ移動させる。
2. 空白 **' '** を出力：文字 **'G'** の上に空白文字を上書きする。
3. 後退 **'\b'** を出力：カーソルを一つ戻して **'F'** の直後へ移動させる。

ただし、出力を命じた結果が、即座に反映されるわけではありません。そこで、**3** では ***fflush*** 関数を呼び出して出力を確実なものとします。

▶ プログラムで出力を指示するたびに画面やファイルなどへ文字を書き込むのでは、高速な出力が行えません。そのため、多くの処理系・環境では、出力すべき文字を“バッファ（一時的なデータの貯蔵庫）”にためておいて、『改行文字の出力が指示された。』とか『バッファが満杯になった。』といったことを契機に実際の出力が行われます。

そのため、プログラムで **"\b \b"** の出力を行っても、これら 3 個の文字がバッファに蓄えられたままとなる可能性があります。文字を確実に消すためには、強制的にバッファの内容をフラッシュする（掃き出させる）必要があります。そこで呼び出すのが ***fflush*** 関数です（この関数の呼出しが不要な環境もあります。しかし、この呼出しを省略すると、文字が即座に消えるかどうかが、環境に依存することになります）。

なお、ストリームやバッファ、さらに ***fflush*** 関数などについては、第 9 章で詳しく学習します。

末尾の文字 **'G'** の消去が完了したら、同じ手順によって、文字 **'F'**, **'E'**, … と後ろから順に 1 文字ずつ消していきます。7 個の文字をすべて消すとプログラムは終了です。

*

2 で関数 *sleep* に渡している値 1000 を変更すると、**文字を消すスピードが変わります**。プログラムを書きかえて試してみましょう。

\r … 復帰

復帰 \r を出力すると、現表示位置が“**その行の先頭**”に移動します。

復帰の働きをうまく利用すれば、画面に表示済みの文字を書きかえることができます。**List 2-3** に示すのは、そのプログラム例です。

まずは実行してみましょう。三つの文字列 "My name is BohYoh." と "How do you do?" と "Thanks." が 2 秒ごとに次々と切りかわって表示されます。

▶ 関数 *sleep* は、前のプログラムのものと同じです。

List 2-3 chap02/return.c

```
/* 復帰\rの利用例：行の書きかえ */

#include <time.h>
#include <stdio.h>

/*--- xミリ秒経過するのを待つ ---*/
int sleep(unsigned long x)
{
    clock_t c1 = clock(), c2;

    do {
        if ((c2 = clock()) == (clock_t)-1)  /* エラー */
            return (0);
    } while (1000.0 * (c2 - c1) / CLOCKS_PER_SEC < x);
    return (1);
}

int main(void)
{
    printf("My name is BohYoh.");
    fflush(stdout);

    sleep(2000);
    printf("\rHow do you do?    ");
    fflush(stdout);

    sleep(2000);
    printf("\rThanks.           ");

    return (0);
}
```

実行結果

```
My name is BohYoh.
```

2秒

```
How do you do?
```

2秒

```
Thanks.
```

カーソルを先頭に戻してから表示

文字を書きかえる原理は単純です。復帰 \r によってカーソルを行の先頭に戻し、そこから別の文字列を表示することによって、同じ行の内容が書きかえられているように**見せかけている**だけです。

なお、2 回目以降に表示する文字列の末尾側を、空白文字で埋めていることに注意しましょう。というのも、**Fig.2-3** に例を示すように、後から出力する文字列のほうが短ければ、もともと表示されていた文字が消えずに残ってしまうからです。

▶ プリンタに対する出力で復帰文字を利用すると、文字の“重ね打ち”ができます。たとえば、文字列 "ABCD\r----" をプリンタに出力すると『ABCĐ』と印刷されます。

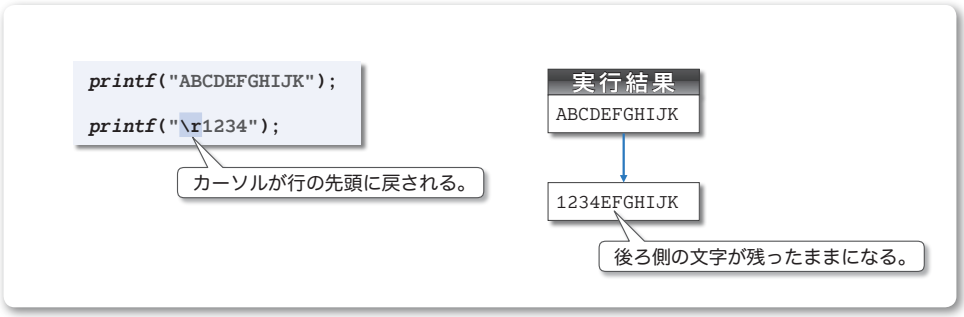


Fig.2-3 復帰\rの働きと注意点

\t … 水平タブ

水平タブ **\t** を出力すると、現表示位置がその行における**“次の水平タブ位置”**に移動します。なお、現表示位置が、行における最後の水平タブ位置にある場合や、その位置を過ぎている場合の動作は規定されていません。

また、水平タブ位置はOSなどの環境に依存します。行の先頭から8桁ごとの間隔の位置に設定されている環境では、**Fig.2-4** の実行結果①が得られます。また、4桁ごとの間隔の位置に設定されている環境では、実行結果②が得られることになります。

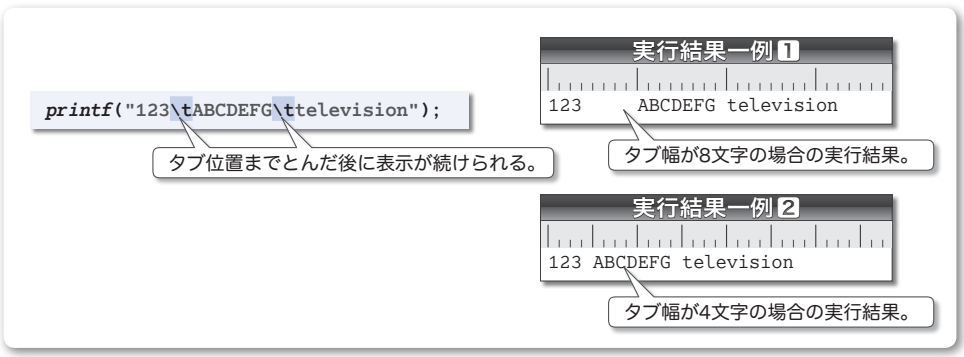


Fig.2-4 水平タブ\tの働き

▶ 第9章では、ファイル内のタブ文字を空白文字に変換したり、その逆の変換を行ったりする便利なユーティリティプログラムを作成します。

\v … 垂直タブ

垂直タブ **\v** を出力すると、現表示位置が**“次の垂直タブ位置における最初の位置”**に移動します。なお、現表示位置が最後の垂直タブ位置にある場合や、その位置を過ぎている場合の動作は規定されていません。

書式送り **\f** と同様に、**主としてプリンタへの出力の際に利用します。**