第１章
まずは慣れよう

もし、物ごとに慣れるだけで上達するのであれば、それに長く親しんだ人ほど“上手（じょうず）”ということになるはずです。ところが、現実はそうではありません。たとえばスポーツを例にとってみても、練習をすればするほど悪いフォームがさらに改悪されていって、どんどん下手になってしまうのを見受けます。プログラミングも慣れるだけでは駄目です。

もっとも、何かを始めようとするときは、まずそのものに実際に触れてみることが不可欠です。本章では、少しだけＣ言語プログラミングをやってみることにしましょう。

1-1　まずは計算結果を表示

整数の和を求めて表示

コンピュータが、電子計算機と呼ばれることからもわかるように、その任務は、何よりも計算をすることです。さっそくC言語を使って、次のような計算を行うことにします。

整数値 15 と 37 の**和**を計算して、その値を表示する。

エディタなどを利用して **List 1-1** をそのまま打ち込みましょう。C言語のプログラムでは、大文字と小文字／全角文字と半角文字は区別されますので、ここに書いてあるとおりにしてください。

List 1-1

```
/*
    整数値15と37の和を表示する
*/

#include  <stdio.h>

int main(void)
{
    printf("%d", 15 + 37);          /* 整数値15と37の和を10進数で表示 */

    return (0);
}
```

実行結果

```
52
```

▶ 本書に示すソースプログラムでは、ある規則にもとづいて、赤文字、*斜体*、**太字**、**ゴシック体**などを使い分けています。これは、読者のみなさんが読みやすいようにとの配慮によるものです。みなさんがプログラムを打ち込む際には、このような区別は不要です。

プログラムとコンパイル

List 1-1 のように、私たち人間が**文字**の並びとして作るプログラムを**ソースプログラム**（*source program*）と呼び、それを格納・保存するファイルを**ソースファイル**（*source file*）と呼びます。

C言語のソースファイルには.cという拡張子を与えるという慣習がありますから、たとえば list0101.c という名前で保存しましょう。

一般に、文字の並びとして作成したソースプログラムは、コンピュータが理解できる形式である**ビット**の並び、すなわち 0 と 1 の並びに変換する必要があり、通常は **Fig.1-1** に示すような翻訳作業を行わねばなりません（ビットに関しては**第 7 章**を参照のこと）。

それらの作業が終わってプログラムを実行すると、画面には 52 と表示されます。

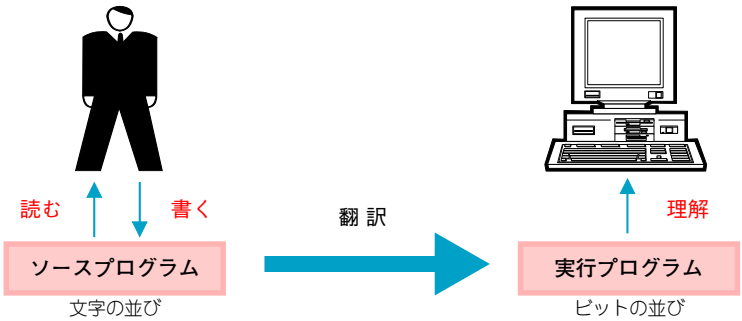

Fig.1-1　ソースプログラムと実行プログラム

▶　翻訳の手順やプログラムの実行方法などは、**処理系**や**実行環境**によって異なりますから、みなさんが利用している処理系のマニュアルなどを参照してください。なお、翻訳や処理系といった言葉については、*p.5* の **Column 1-1** で説明しています。

ソースプログラム中に綴り間違いなどがあると、翻訳時にエラーが発生し、その旨を伝える**診断メッセージ**が表示されます。そのようなときは、打ち込んだプログラムをよく読み直してミスを取り除いた上で、再度コンパイルを試みてください。

プログラムには # や { などの記号が多いので、少しとまどったかもしれませんね。でも大丈夫。少しずつ理解していきましょう。

▶　記号文字の読み方は、*p.7* にまとめています。

注釈

ソースプログラムの赤で示した部分、すなわち /* から */ までは、**注釈**（*comment*）です。注釈の有無や、その内容によって、プログラムの動作が変わるといったことは **ありません**。プログラムの作成者を含めて、その読み手に伝えたいことを、日本語や英語などの簡潔な言葉で書き込んでおけば、読みやすさがグンと向上します。

▶　**List 1-1**くらい短いプログラムはともかく、他人の作った長いプログラムを読んで理解するのは、とても大変です。また、自分が作ったプログラムのすべてを永久に記憶しておくことは不可能ですから、プログラム作成者自身にとっても、適切な注釈を書き残しておくことは重要です。

■ 重 要 ■

ソースプログラムには、読み手に伝えたいことがらを、注釈として簡潔に記述せよ。

プログラムからもわかるように、注釈は、複数の行にまたがることができます。なお、注釈を閉じるための記号を間違えて /* としないでください。どこまでも注釈とみなされることになってしまいます。

おまじない

プログラムから注釈を取りさると、**Fig.1-2**のようになります。水色の部分は、“**おまじない**”であると心得(こころえ)てください。これらの意味は、後の章で少しずつ解説していきます。その綴(つづ)りを含めて、全てを丸暗記しましょう。

当分は、“おまじない”の部分はそのまま利用して、そうでないところだけを自分で作ることにします。

```
#include  <stdio.h>

int main(void)
{
    printf("%d", 15 + 37);

    return (0);
}
```

Fig.1-2　プログラムとおまじない

printf … 書式化して表示を行う関数

表示を行うのが ***printf*** という**関数**（*function*）です（一般に ***printf*** はプリントエフなどと呼ばれます。末尾の f は書式という意味の *format* に由来します）。

関数を利用するには、**関数呼出し**（*function call*）を行わねばなりません。***printf*** 関数を呼び出して 15 と 37 の和を表示する部分は **Fig.1-3** のように解釈できます。

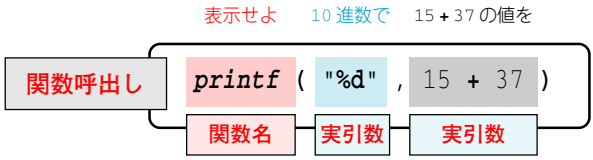

Fig.1-3　printf 関数の呼出し

この関数呼出しは「表示してください」という依頼であり、その補助的な指示が ( ) の中に与えられる**実引数**（*argument*）です。なお、この例のように実引数が二つ以上あるときは、コンマ , で区切ることになっています。

さて、***printf*** 関数に与えている最初の実引数 **"%d"** は、

続いて与える実引数の値を 10 進数で表示してください。

と、出力の書式を指示するものです。したがって、この呼出しによって、二つめの実引数である 15 + 37 の値、すなわち 15 と 37 の**和**が、52 と表示されることになります。

▶ **"%d"** の d は、10 進数という意味の *decimal* に由来します。10 進数や 2 進数などの数については *p.149* で、8 進数や 16 進数での表示については *p.170* で説明します。
また、***printf*** 関数の詳細は *p.318* にまとめています。

■ 重 要 ■

関数呼出しは、処理を行ってもらうための依頼のようなものであり、その際に必要な補助的な指示は、**実引数**として与える。

文

プログラムをよく見てください。***printf***関数の呼出しにはセミコロン　**;**　が付いていますし、おまじないである

```
return (0);
```

にもセミコロンが付いています。これは、日本語での句点　。に相当するものです。

最後に句点があって、日本語として正しい文となるように、C言語でも、セミコロンを与えることによって正しい**文**（*statement*）となるのです。

■ 重 要 ■

文の末尾には、原則としてセミコロン；が必要である。

整数の差を求めて表示

15から37を引いた値を表示するプログラムを**List 1-2**に示します。

List 1-2

```
/*
    整数値15から37を引いた値を表示する
*/

#include  <stdio.h>

int main(void)
{
    printf("%d", 15 - 37);       /* 整数値15から37を引いた値を10進数で表示 */

    return (0);
}
```

実 行 結 果

```
-22
```

プログラムを実行すると、-22と表示されます。計算結果が負の値となったときには、マイナス記号がちゃんと表示されることがわかりますね。

Column 1-1　翻訳フェーズとコンパイル

C言語のプログラムを実行させるには、理論上は8段階もの**翻訳フェーズ**（*translation phase*）を経ることになっています。なお、ソースプログラムを実行させるために必要なソフトウェアのことを**処理系**（*implementation*）と呼びます。

C言語の処理系では、多くの場合、ソースプログラムをコンピュータが直接理解・実行できる形式に“翻訳”する**コンパイル方式**（本文で解説した方式です）が採用されていますが、プログラムを1行ずつ解釈しながら実行する**インタプリタ方式**（実行速度は遅くなる傾向にあります）などもあります。

書式文字列と変換指定

プログラムを実行したときに、ただ和や差の値だけが表示されても、何のことだかわかりませんから、もう少し親切に表示することにします。そのプログラムが **List 1-3** です。

今回は、***printf*** 関数に与えている最初の実引数が、長く複雑になっています。

List 1-3

```
/*
    整数値15と37の和を親切に表示
*/

#include  <stdio.h>

int main(void)
{
    printf("15と37の和は%dです。\n", 15 + 37);      /* 表示後に改行 */

    return (0);
}
```

実 行 結 果

```
15と37の和は52です。
```

これまで説明していませんでしたが、プログラムの水色部分、すなわち***printf*** 関数に与える最初の実引数を、**書式文字列**（*format string*）と呼びます。

書式文字列の **%d** の部分は、**続く実引数を 10 進数で表示せよ**という書式を指示するための**変換指定**（*conversion specification*）です。書式文字列中の変換指定でない文字は、（基本的には）そのまま出力されます。

書式文字列の末尾の **\n** は、**改行**（*new line*）を表すための特別な表記です。\ と n を組み合わせて“改行文字”という特殊な一つの文字を表します。

▶　日本で多くのパソコンに採用されている JIS コードという文字体系（*p.200*）では、逆斜線 \ の代わりに、円記号 ¥ を使います。したがって、みなさんの環境によっては、¥ を使わなければならないかもしれません。その場合は、本書のすべての \ を ¥ と読みかえてください。なお、最後に改行文字が必要である理由は右ページ **Column 1-2** を参照してください。

本プログラムによる画面への出力の様子を示したのが **Fig.1-4** です。

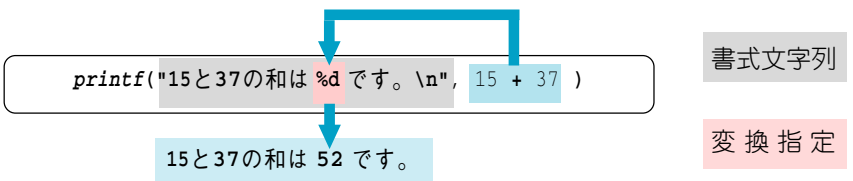

Fig.1-4　書式文字列と変換指定

● 演 習 1 - 1

15から37を引いた値を計算して「15から37を引いた値は-22です。」と表示するプログラムを作成せよ。

記号文字の読み方

C言語で使う記号文字の読み方を、俗称も含めてまとめたものが**Table 1-1**です。

Table 1-1　記号文字の読み方

+	プラス符号、正符号、プラス、たす
-	マイナス符号、負符号、ハイフン、マイナス、ひく
*	アステリスク、アスタリスク、アスター、かけ、こめ、ほし
/	スラッシュ、スラ、わる
\	逆斜線、バックスラッシュ、バック　※ JISコードでは¥
¥	円記号、円、円マーク
%	パーセント
.	ピリオド、小数点文字、ドット、てん
,	コンマ、カンマ
:	コロン、ダブルドット
;	セミコロン
'	一重引用符、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
(	左括弧、左丸括弧、左小括弧、パーレン
)	右括弧、右丸括弧、右小括弧
{	左波括弧、左中括弧、ブレイス
}	右波括弧、右中括弧
[	左角括弧、左大括弧、ブラケット
]	右角括弧、右大括弧
<	小なり
>	大なり
?	疑問符、はてな、クエッション、クエスチョン
!	感嘆符、エクスクラメーション、びっくりマーク、びっくり、ノット
&	アンド、アンパサンド
~	チルダ、なみ、にょろ　※ JISコードでは‾（オーバライン）
^	アクサンシルコンフレックス、ハット
#	シャープ、ナンバー
_	下線、アンダーライン、アンダーバー、アンダースコア
=	等号、イクオール
|	縦線

Column 1-2　改行の必要性

List 1-1の実行プログラム名が`list0101`であるとします。多くの実行環境では、プログラムを実行すると、左下図のように、プログラムの出力結果である52の直後にプロンプトがくっついてしまいます（▷は、オペレーティングシステムのプロンプトであり、MS-DOSであれば**A>**などの記号が、UNIXであれば**%**などの記号が表示されます）。したがって、**List 1-3**のように、プログラムの最後の出力には改行を出力したほうがよいのです（右下図）。

```
▷ list0101↵
52 ▷
```

```
▷ list0103↵
15 と 37 の和は 52 です。
▷
```

書式化を行わない表示

実引数を一つだけ与えて***printf***関数を呼び出すこともできます。そのときは、書式文字列内の文字が、そのまま表示されます。『こんにちは。私の名前は柴田望洋です。』と表示するプログラムを**List 1-4**に示します。

List 1-4

```
/*
    挨拶をして自己紹介をする
*/

#include  <stdio.h>

int main(void)
{
    printf("こんにちは。私の名前は柴田望洋です。\n");

    return (0);
}
```

実行結果

```
こんにちは。私の名前は柴田望洋です。
```

このプログラムを少し変更して、『こんにちは。』と『私の名前は柴田望洋です。』とを別々の行に表示するように書きかえたプログラムが**List 1-5**です。

List 1-5

```
/*
    挨拶をして自己紹介をする（挨拶と自己紹介を別の行に表示）
*/

#include  <stdio.h>

int main(void)
{
    printf("こんにちは。\n私の名前は柴田望洋です。\n");  /* 途中で改行 */

    return (0);
}
```

実行結果

```
こんにちは。
私の名前は柴田望洋です。
```

書式文字列の途中に書かれた**\n**によって改行が行われるのですね。**List 1-6**に示すように、***printf***関数の呼出しを二つに分けても同じ結果が得られます。

List 1-6

```
/*
    挨拶をして自己紹介をする（挨拶と自己紹介を別々に表示）
*/

#include  <stdio.h>

int main(void)
{
    printf("こんにちは。\n");                /* 表示後に改行 */
    printf("私の名前は柴田望洋です。\n");    /* 表示後に改行 */

    return (0);
}
```

実行結果

```
こんにちは。
私の名前は柴田望洋です。
```

文字列リテラル

"ABC"や"こんにちは。"のように、一連の文字を二重引用符"で囲んだものは、ひと続きの文字の並びを表し、**文字列リテラル**(*string literal*)と呼ばれることを覚えておきましょう。

▶　本来、文字列リテラルの中に漢字などの全角文字を使うのは、規則違反です。もっとも、私たちが日本で使う処理系の大部分は、全角文字が利用できるようになっています。読者のみなさんが読みやすいようにとの配慮から、本書では全角文字を使っています。

拡張表記

改行を表すための特別な表記が**\n**であることは既に説明しましたが、このような特別な表記を**拡張表記**（エスケープシーケンス）（*escape sequence*）と呼びます。

警報（*alert*）を発する拡張表記が**\a**です。『こんにちは。』と表示して、警報を3回発するプログラムを**List 1-7**に示します。

List 1-7

```
/*
    警報を3回発する
*/

#include  <stdio.h>

int main(void)
{
    printf("こんにちは。\a\a\a\n");      /* 表示とともに警報を3回発する */

    return (0);
}
```

実行結果

```
こんにちは。
```

▶　プログラムを実行する環境によっては、警報（音でなく視覚的なものである場合もありますが、普通はいわゆる《ビープ音》です）がならないことや、三つの警報が続けてなることもあります。

● 演習 1-2

右に示すような表示を行うプログラムを作成せよ。ただし、プログラム中、***printf***関数の呼出しは、1回限りとする。

```
風
林
火
山
```

● 演習 1-3

右に示すような表示を行うプログラムを作成せよ。ただし、プログラム中、***printf***関数の呼出しは、1回限りとする。

```
もしもし。
こんにちは。

それでは。
```

1-2　変数

定数と変数

ここまでは、プログラム中に埋め込まれた値である**定数**（*constant*）の和や差を求めて表示するプログラムばかりでした。もう少し複雑な計算になると、その途中結果を覚えさせたりするために、**変数**を使う必要が生じます。

『変数』という言葉を聞くと、数学が嫌いな人は、中学や高校での方程式などを思い出してイヤな感じになるかもしれませんが、心配は無用です。次のように考えましょう。

■ 重 要 ■

変数とは、数値などを格納するための『箱』である。

数値などを格納する魔法の箱（!!）を使うには、それなりの手続きが要求され、

```
int  vx;
```

といった**宣言**（*declaration*）を事前に行わねばなりません（一般に`int`はイントと呼ばれます）。**Fig.1-5** にも示していますが、この宣言によって、*vx* という名前の箱が一つ用意されることになります。

この箱には、整数値のみを格納することができ、*vx* は“**`int` 型**”であると呼ばれます。

▶ `int` は、整数という意味の *integer* に由来します。

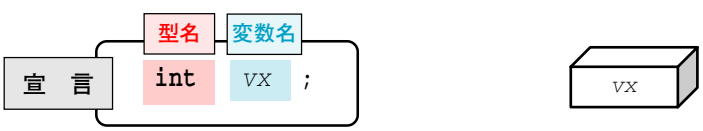

Fig.1-5　変数と宣言

この例での変数名は*vx*ですが、自由に名前を与えることができます。たとえば*i*や*x*のように、1 文字だけの名前でも構いません。

▶ 命名の規則は、*p.82* に示します。

実際に変数を使ったプログラムを作成しましょう。次の問題を考えます。

変数に適当な値を代入して、その値を表示する。

このように作成したプログラムの例を **List 1-8** に示します。

List 1-8

```
/*
    二つの変数に整数値を格納して表示
*/

#include  <stdio.h>

int main(void)
{
    int  vx, vy;                        /* vxとvyはint型の変数 */

    vx = 57;                            /* vxに57を代入 */
    vy = vx + 10;                       /* vyにvx+10を代入 */

    printf("vxの値は%dです。\n", vx);    /* vxの値を表示 */
    printf("vyの値は%dです。\n", vy);    /* vyの値を表示 */

    return (0);
}
```

実行結果

```
vxの値は57です。
vyの値は67です。
```


複数の変数を宣言

二つの変数 *vx* と *vy* は、コンマ　,　で区切って宣言されています。これで、*vx* という名前の変数と *vy* という名前の変数が用意されることになります。

なお右に示すように、二つの変数を個別に宣言しても構いません。

```
int  vx;
int  vy;
```

▶　各行に一つずつ宣言を書くことによって、その宣言に対する注釈を記入しやすくなりますし、宣言の追加や削除もスムーズに行えるようになります。ただし、プログラムの行数は増えてしまいますね。時と場合に応じて臨機応変に使い分けましょう。

なお、このプログラムでは、宣言の次の行などが、何も書かれておらず空(あ)けられています。このような、ちょっとした工夫で、プログラムは読みやすくなります。

代入

このプログラムで初登場の記号 = は、**右側の値を左側に代入しなさい**という意味です。したがって、まず変数 *vx* に 57 が代入されます（**Fig.1-6**）。

```
vx = 57;
vy = vx + 10;
```

なお、変数の値はいつでも取り出せるようになっています。*vx*の値を取り出したものに 10 を加えた値を *vy* に代入しますので、*vy* の値は 67 となります。

もちろん、文の末尾には、セミコロン ; を忘れてはいけません。

▶　数学のように、『*vx* と 57 が等しい』といっているのではないことに注意しましょう。

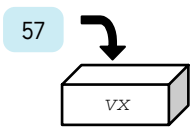

Fig.1-6　変数への値の代入

1-3　読込みと表示

キーボードからの読込み

画面に表示を行うだけでは面白くありませんので、キーボードから値を読み込んで、もっと対話的にしましょう。

整数値を読み込んで、その値を表示して確認する。

プログラムを **List 1-9** に示します。

List 1-9

```
/*
    読み込んだ整数の値を表示して確認
*/

#include  <stdio.h>

int main(void)
{
    int  no;

    printf("整数を入力してください：");
    scanf("%d", &no);                          /* 整数値を読み込む */

    printf("あなたは%dと入力しましたね。\n", no);

    return (0);
}
```

実 行 例

```
整数を入力してください：37
あなたは37と入力しましたね。
```

▶　実行例での赤文字は、入力の一例であり、それによって表示が異なる数値などを**太字**で示しています。53 と入力すれば、『あなたは 53 と入力しましたね。』と表示されます。

scanf…読込みを行う関数

キーボードから数値などを読み込むために用いるのが ***scanf*** 関数です（一般に ***scanf*** はスキャンエフなどと呼ばれます）。

ここでの変換指定 "%d" は、***printf*** の場合と同様であり、10 進数の指定です。したがって、

```
scanf("%d", &no);
```

キーボードから 10 進数を読み込んで、その値を *no* に格納してください。

と依頼していることになります（**Fig.1-7**）。

ただし、***printf***の場合とは異なり、変数名の前に**&**という奇妙な記号を付けなければならないことに注意しましょう。

▶　**&** の具体的な意味などは、**第 10 章**（*p.236*）で解説します。

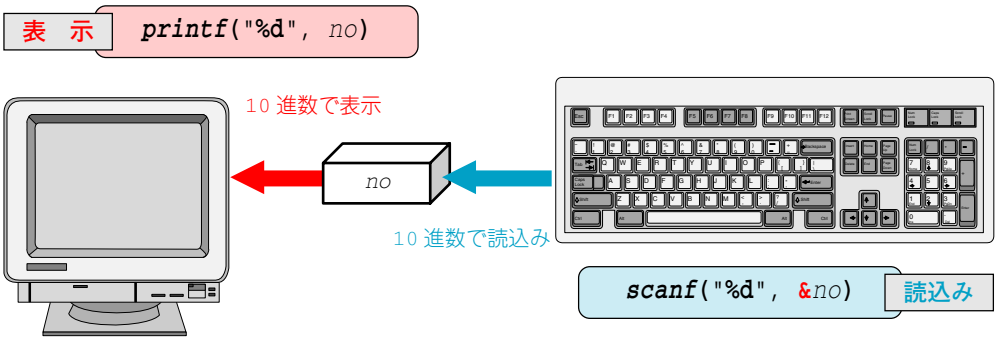

Fig.1-7　printf 関数による表示と scanf 関数による読込み

積を求める

整数値を読み込んで、その 10 倍の値を表示するプログラムが **List 1-10** です。

List 1-10

```
/*
    読み込んだ整数の10倍の値を表示
*/

#include  <stdio.h>

int main(void)
{
    int  no;

    printf("整数を入力してください：");
    scanf("%d", &no);

    printf("その数の10倍は%dです。\n", 10 * no);

    return (0);
}
```

実 行 例

```
整数を入力してください：357↵
その数の10倍は3570です。
```

ここで初めて利用した記号*は、掛け算すなわち乗算を行うためのものです。×ではなく * であることに注意しましょう。もちろんプログラム中の 10 * *no* は、*no* * 10 でも構いません。

● 演 習 1-4

右に示すように、読み込んだ整数値に10を加えた値を表示するプログラムを作成せよ。

```
整数を入力してください：57↵
その数に10を加えると67です。
```

● 演 習 1-5

右に示すように、読み込んだ整数値から10を減じた値を表示するプログラムを作成せよ。

```
整数を入力してください：57↵
その数から10を減じると47です。
```

puts…表示を行う関数

変数を利用して、もう少しだけ難しい問題を解きましょう。

二つの整数値を読み込んで、その和を表示する。

1

このプログラムを **List 1-11** に示します。

List 1-11

```
/*
    読み込んだ二つの整数値の和を表示
*/

#include  <stdio.h>

int main(void)
{
    int  n1, n2;

    puts("二つの整数を入力してください。");
    printf("整数１：");    scanf("%d", &n1);
    printf("整数２：");    scanf("%d", &n2);

    printf("それらの和は%dです。\n", n1 + n2);        /* 和を表示 */

    return (0);
}
```

実 行 例

```
二つの整数を入力してください。
整数１：27↵
整数２：35↵
それらの和は62です。
```

▶　このプログラムのように、C言語では、行の中に複数の文を書けます（一つの文が複数の行にまたがっても構いません）。プログラムの表記に関しては *p.84* で解説します。

本プログラムで初めて使用したのが***puts***関数です（末尾のsは*string*に由来し、一般に***puts***はプットエスなどと呼ばれます）。

puts関数は、実引数として与えられた文字の並びを出力して、さらに最後に改行を行います。すなわち、**Fig.1-8**にも示すように、***puts*("…")**は、***printf*("…\n")**とほぼ同じ働きをします。

書式化の必要がなくて、改行もしたいという場合は、***printf***ではなくて***puts***の方を使用しましょう。

▶　***puts*** 関数に与えることのできる実引数は一つだけであることに注意しましょう。

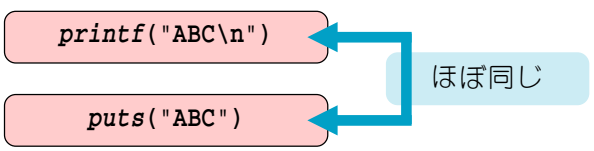

Fig.1-8　printf 関数と puts 関数

このプログラムを少し書きかえたのが **List 1-12** です。読み込んだ整数値の和を変数*wa*にいったん格納しておき、その値を表示します。もちろん、プログラムの見かけ上の動作は、**List 1-11** と同じです。

List 1-12

```
/*
    読み込んだ二つの整数値の和を求めて表示
*/

#include  <stdio.h>

int main(void)
{
    int  n1, n2;
    int  wa;          /* 和 */

    puts("二つの整数を入力してください。");
    printf("整数１：");    scanf("%d", &n1);
    printf("整数２：");    scanf("%d", &n2);

    wa = n1 + n2;                          /* n1とn2の和をwaに代入 */

    printf("それらの和は%dです。\n", wa);      /* 和を表示 */

    return (0);
}
```

実行例

```
二つの整数を入力してください。
整数１：27↵
整数２：35↵
それらの和は62です。
```

● 演習 1-6

右に示すような表示を行うプログラムを作成せよ。ただし、表示には***printf*** 関数ではなく***puts*** 関数を利用すること。

```
風
林
火
山
```

● 演習 1-7

右に示すように、読み込んだ二つの整数値の積を表示するプログラムを作成せよ。

```
二つの整数を入力してください。
整数１：27↵
整数２：35↵
それらの積は945です。
```

● 演習 1-8

右に示すように、読み込んだ三つの整数値の和を表示するプログラムを作成せよ。

```
三つの整数を入力してください。
整数１：7↵
整数２：15↵
整数３：23↵
それらの和は45です。
```