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）”になるはずです。ところが、実際は、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（へた）”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（だめ）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す。

1–1 まずは表示を行う

コンピュータで計算を行っても、画面に表示しなければ、計算結果がわかりません。本節では、画面への表示方法を学習します。

整数の加算の結果を表示

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“計算を行う”ことです。さっそくC言語を使って、次の計算を行います。

整数値 15 と 37 を加算して、その値を表示する。

エディタなどを利用して **List 1-1** を打ち込みましょう。なお、プログラム中の大文字と小文字、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ます。

List 1-1 chap01/list0101.c

```
/*
    整数値15と37を加えた結果を表示
*/                                  studio と間違えないように !!
#include <stdio.h>

int main(void)
{
    printf("%d", 15 + 37);          /* 整数値15と37を加えた結果を10進数で表示 */

    return 0;
}                                   余白はタブキーあるいはスペースキーで打ち込む（全角文字のスペースは不可）。
```

実行結果

```
52
```

▶ プログラム中の余白や " などの記号を全角文字で打ち込んではいけません。余白の部分は、スペースキーかタブキーを使って打ち込みます（p.105 で詳しく学習します）。

なお、本書に示すプログラムは、ホームページからダウンロードできます（p.v）。各プログラムリストの右上に示しているのは、フォルダ名を含むファイル名です。

プログラムとコンパイル

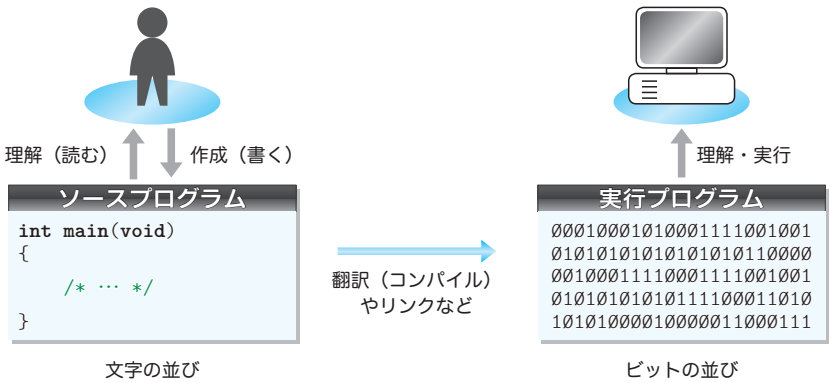
私たちが“文字の並び”として作るプログラムを**ソースプログラム**（*source program*）と呼び、それを格納したファイルを**ソースファイル**（*source file*）と呼びます。

▶ source は、“もとになるもの”という意味です。そのため、ソースプログラムは、**原始プログラム**と呼ばれることもあります。

C言語のソースファイルには .c という拡張子（かくちょうし）を与える慣習がありますので、たとえば chap01 といった名前のフォルダの中に、list0101.c という名前で保存します。

＊

さて、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ます。



● **Fig.1-1** ソースプログラムと実行プログラム

Fig.1-1 に示すように、ソースプログラムを翻訳（ほんやく）＝**コンパイル**したり**リンク**したりする作業を行って、**実行プログラム**を作成します。

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 52 と表示されます。

▶ 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（p.370）。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** で学習します。

ソースプログラムに綴（つづ）り間違いなどがあると、翻訳時にエラーが発生して、翻訳を行う**コンパイラ**が、その旨の**診断メッセージ**（*dignostic message*）を表示します。そのときは、打ち込んだプログラムのミスを取り除いた上で、再度コンパイルを試みます。

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。でも大丈夫です。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

ソースプログラム中の /* から */ までは、注釈（コメント）（*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。

重要 ソースプログラムには、プログラム作成者自身を含めた読み手に伝えるべきことがらを、注釈として簡潔に記述せよ。

注釈は、複数行にまたがれます。なお、注釈を閉じるための記号を /* に間違えると、/* からプログラムの最後までが注釈とみなされてしまいます。

おまじない

プログラムから注釈を取りさったものを、**Fig.1-2** に示しています。しばらくは、水色の部分を“おまじない”と考えることにします。これらの意味は、後の章で少しずつ学習していきますので、各部の綴りを含めて、丸暗記しておきましょう。

当分は、“おまじない”の部分はそのまま利用して、そうでないところを自分で作っていきます。

```
#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

▶ stdio は、standard I/O（標準入出力）の略です。studio と間違えないようにします。

printf 関数：書式化して表示を行う関数

画面への表示を行うのが ***printf*** という**関数**（*function*）です（***printf*** はプリントフでは**なく**、プリントエフと発音します。末尾の f は書式という意味の format に由来します）。

関数を利用するために行うのが、**関数呼出し**（*function call*）です。***printf*** 関数を呼び出して 15 と 37 の和を表示する箇所は、**Fig.1-3** のように解釈できます。

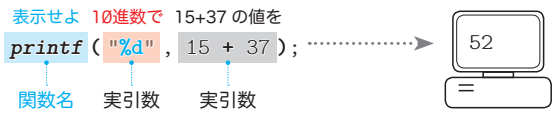


Fig.1-3 printf 関数の呼出しによる画面への表示

関数呼出しは、「処理の依頼」です。そして、その際の「補助的な指示」が、( ) の中に与える**実引数**（*argument*）です。この例のように実引数が二つ以上あるときは、コンマ , で区切ります。

さて、先頭の実引数 "%d" は、

> 続いて与える実引数の値の表示を“**10 進数**”で行ってください。

という書式の指示です。この指示によって、二つ目の実引数である 15 + 37 の値が 10 進数で 52 と表示されます。

▶ "%d" の d は、10 進数という意味の語句 decimal に由来します。10 進数以外の数や表示などは、第 7 章で学習します。また、***printf*** 関数の詳細は、p.354 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であり、その際に必要な補助的な指示は、( ) の中に実引数として与える。

文

printf 関数の呼出しにはセミコロン；が付いています。また、おまじないの“return 0;”にもセミコロンが付いています。これは、日本語の句点（くてん）。に相当するものです。

末尾に句点があって、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*）となります。

重要 文の末尾には、原則としてセミコロン；が必要である。

プログラムを実行すると、おまじない中の{と}のあいだに置かれた文が順次実行される仕組みとなっています（詳細は第6章で学習します）。

整数の減算の結果を表示

加算のプログラムを、減算（引き算）に変更するのは容易です。15から37を減じた（引いた）値を表示するプログラムを **List 1-2** に示します。

▶ **List 1-1** をコピーして、違うところのみを変更すると素早くプログラムを作れます。

List 1-2 chap01/list0102.c

```
/*
    整数値15から37を減じた結果を表示
*/

#include <stdio.h>

int main(void)
{
    printf("%d", 15 - 37);  /* 整数値15から37を減じた結果を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。このように、演算結果が負となったときには、先頭にマイナス記号が表示されます。

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（たとえば、Visual C++などの処理系があります）。

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演算結果だけが表示されても、何のことだかわかりませんので、もう少し丁寧に表示しましょう。そのプログラムが **List 1-3** です。

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

List 1-3 chap01/list0103.c

```
/*
    整数値15と37を加えた結果を丁寧に表示
*/

#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;
}
```

文字 \ については、必ず右ページを参照のこと !!

実行結果

```
15と37の和は52です。
```

プログラムの網かけ部（**Fig.1-4** の点線で囲まれた箇所）、すなわち ***printf*** 関数に与える最初の実引数は、**書式文字列**（*format string*）と呼ばれます。

書式文字列中の **%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**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。

なお、本プログラムの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。2 個の文字 \ と n を組み合わせて、“改行文字”という 1 個の文字を表します。

▶ 画面に \ と n の 2 文字が表示されるのではなく、（目に見えない）改行文字が出力されます。

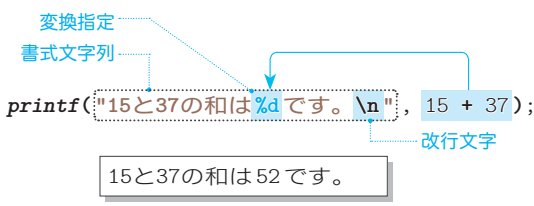


Fig.1-4 書式文字列と変換指定と改行文字

Column 1-2 プログラム最後の表示における改行の必要性

右に示すのは、**List 1-1** の実行の様子です（▷は、オペレーティングシステムのプロンプトであり、> や % などの記号が表示されます）。

多くの実行環境では、プログラムを実行すると、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。

List 1-3 のように、プログラムの最後に改行文字を出力しておけば、プロンプトが続くことはありません（下図）。

```
▷list0101↵
52▷
```

```
▷list0103↵
15と37の和は52です。
▷
```

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

記号	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、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、バックスラ、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
'	単一引用符、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、開き括弧、左丸括弧、始め丸括弧、左小括弧、始め小括弧、左パーレン
)	右括弧、閉じ括弧、右丸括弧、終り丸括弧、右小括弧、終り小括弧、右パーレン
{	左波括弧、左中括弧、始め中括弧、左ブレイス、左カーリーブラケット、左カール
}	右波括弧、右中括弧、終り中括弧、右ブレイス、右カーリーブラケット、右カール
[	左角括弧、始め角括弧、左大括弧、始め大括弧、左ブラケット
]	右角括弧、終り角括弧、右大括弧、終り大括弧、右ブラケット
<	小なり、左アングル括弧、左向き不等号
>	大なり、右アングル括弧、右向き不等号
?	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、チルド、なみ、にょろ　※JISコードでは￣（オーバライン）
￣	オーバライン、上線、アッパライン
^	アクサンシルコンフレックス、ハット、カレット、キャレット
#	シャープ、ナンバー
_	下線、アンダライン、アンダバー、アンダスコア
=	等号、イクオール、イコール
|	縦線

注意!!

▶ 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では、逆斜線記号文字\の代わりに、円記号文字¥を使います。もし、**みなさんの環境が¥を使う環境であれば、本書のすべての\を¥と読みかえてください。**

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。

書式化を行わない表示

実引数を一つだけ与えて *printf* 関数を呼び出すと、書式文字列の文字がそのまま表示されます。**List 1-4** で確認しましょう。これは、『こんにちは。私の名前は＊＊＊＊です。』と表示するプログラムです。

▶ みなさんは、ご自身の名前に変更してプログラムを打ち込むようにしましょう。

List 1-4 chap01/list0104.c

```
/*
    挨拶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");    /* 表示後に改行 */

    return 0;
}
```

自分の名前に変更すること !!

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しましょう。それが **List 1-5** のプログラムです。

List 1-5 chap01/list0105.c

```
/*
    挨拶して自己紹介をする（挨拶と自己紹介を別の行に表示・その１）
*/

#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"); /* 途中と最後で改行 */

    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ます。なお、**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 chap01/list0106.c

```
/*
    挨拶して自己紹介をする（挨拶と自己紹介を別々に表示・その２）
*/

#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。
```

▶ こちらのほうが、プログラムが読みやすいですね。

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、文字の並びを表します。**文字列リテラル**（*string literal*）と呼ばれます。

▶ リテラルとは、『文字どおりの』『文字で表された』という意味です。本書では、文字列リテラルを "少し薄い色の文字" で表記します。なお、本来、文字列リテラルの中で漢字などの全角文字を使うことには、若干（じゃっかん）の問題があります。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ます。読者のみなさんが理解しやすいように、との配慮から、本書では文字列リテラル中に全角文字を多用しています。

拡張表記

改行文字を表すための特別な表記が **\n** であることは、既に学習しました。このような特別な表記は、**拡張表記**（*escape sequence*）と呼ばれます。

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 chap01/list0107.c

```
/*
    挨拶して警報を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 */
                                  \a は警報で、\n は改行。
    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）が鳴らないことや、三つの警報が一つにまとまって鳴ることもあります。

なお、本書の実行結果では、警報を♪と表記します。

演習 1–2

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

```
天
地
人
```

演習 1–3

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

```
もしもし。

こんにちは。
それでは。
```

1–2 変数

計算の途中結果や最終的な結果を覚えさせておくために利用するのが、変数です。本節では、変数の使い方を学習します。

変数と宣言

ここまでは、プログラムに埋め込まれた値である**定数**（*constant*）の加算や減算の結果を表示するプログラムが中心でした。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**（*variable*）を使う必要が生じます。

数学が嫌いな人は、『変数』という言葉を聞くと、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。

変数とは、数値や文字などを格納するための『箱』である。

数値を格納する魔法の箱!!である変数に値を入れておくと、その箱が存在する限り、値が保持されます。また、値を取り出したり書きかえたりするのも自由です。

変数を使うには、それなりの手続きが要求されます。以下の**宣言**（*declaration*）が必要です（一般に **int** はイントと呼ばれます）。

```
int n;        /* 型がintで名前がnの変数の宣言 */
```

Fig.1-5 に示すように、この宣言によって、*n* という名前の変数（箱）が用意されます。この変数には整数値のみを格納でき、変数 *n* は“**int** 型”であると呼ばれます。

▶ **int** は、整数という意味の語句 integer に由来します。また、『型』については、主に第 2 章と第 7 章で詳しく学習します。

この例での変数名は *n* ですが、与える名前は自由です。たとえば、*i* や *no* や *year* のように、文字数も（ある程度）自由です。

▶ 命名の規則は、p.102 で学習します。

重要 変数を使うには、その型と名前を宣言によって明確にしなければならない。

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。

二つの変数に適当な値を代入して、その値を表示する。

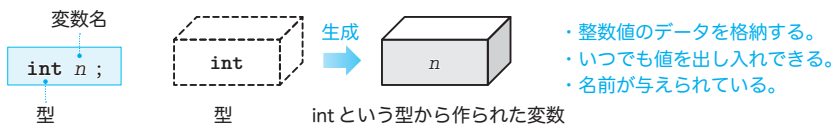


Fig.1-5 変数

作成したプログラムを **List 1-8** に示します。

List 1-8 chap01/list0108.c

```
/*
    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 */

1   vx = 57;                            /* vxに57を代入 */
2   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。
```

二つの **int** 型変数 *vx* と *vy* が、コンマ記号 , で区切って宣言されています。

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;  /* 変数（その１）*/
int vy;  /* 変数（その２）*/
```

▶ 各行に一つずつ宣言を書くことによって、各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、プログラムの行数が増えますので、臨機応変（りんきおうへん）に使い分けましょう。

本プログラムでは、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。このような少しの気配（きくば）りで、プログラムは読みやすくなります。

代入

本プログラムで初登場の記号 = は、『右の値を左側の変数に代入せよ。』という指示です。そのため、1では変数 *vx* に 57 が代入されます（**Fig.1-6**）。

▶ 数学のように、『*vx* と 57 が等しい。』といっているのではありません。

変数の値はいつでも取り出せますので、2では、*vx* の値を取り出したものに 10 を加えた値を *vy* に代入しています。

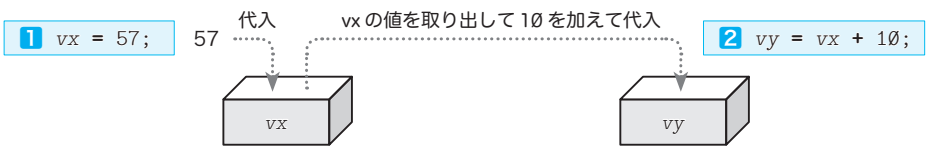


Fig.1-6 変数への値の代入と取出し

初期化

前ページのプログラムから、変数に値を代入する部分を削除するとどうなるかを実験しましょう。**List 1-9** を実行してみてください。

List 1-9 chap01/list0109.c

```
/*
    二つの変数に値を代入せずに表示
*/
#include <stdio.h>

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

    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;
}
```

実行結果一例

```
vxの値は3535です。
vyの値は938です。
```

変数 vx と vy が妙な値になっています。変数が生成される際は、**不定値**すなわち**ゴミの値**が入れられるからです（**Fig.1-7**）。

そのため、値が設定されていない変数から値を取り出すと、思いもよらぬ結果となります。

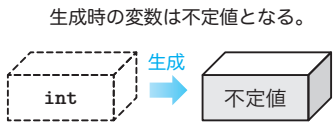


Fig.1-7 生成時の変数の値

▶ 表示される値は、実行環境や処理系によって異なります（実行時エラーが発生して、プログラムの実行が中断される場合もあります）。また、同一環境であっても、プログラムを実行するたびに異なる値が表示される可能性があります。

なお、静的記憶域期間をもつ変数に限り、生成時に 0 が入れられます。第 6 章 (p.162) で学習します。

初期化を伴う宣言

変数に入れる値が事前にわかっているのであれば、その値を最初から変数に入れておくべきです。そのように修正したプログラムが **List 1-10** です。

網かけ部の宣言によって、変数 vx と vy は、それぞれ 57 と vx + 10（すなわち 67）とで**初期化**（*initialize*）されます。変数の宣言における = 記号以降の右側の部分は、変数の生成時に入れる値を指定するものであり、**初期化子**（*initializer*）と呼ばれます（**Fig.1-8** a）。

整数を格納する箱である変数を作る際に、そこに入れるべき値が事前にわかっていれば、最初から値を入れておいたほうが自然です。

重要 変数は生成されたときに不定値が入れられる。そのため、変数を宣言する際は、特に不要でない限り、必ず初期化を行うべきである。

List 1-10 chap01/list0110.c

```
/*
    二つの変数を初期化して表示
*/

#include <stdio.h>

int main(void)
{
    int vx = 57;                    /* vxはint型の変数（57で初期化）*/
    int vy = vx + 10;               /* vyはint型の変数（vx + 10で初期化）*/

    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。
```

初期化と代入

本プログラムで行っている初期化と、**List 1-8**（p.11）で行った代入は、値を入れるタイミングが異なります。以下のように理解しましょう（**Fig.1-8**）。

- **初期化**：変数を生成するときに値を入れること。
- **代　入**：生成ずみの変数に値を入れること。

▶ 本書では、初期化を指示する記号を細字の = で示し、代入を指示する記号を**太字の =** で示すことによって区別しやすくしています。

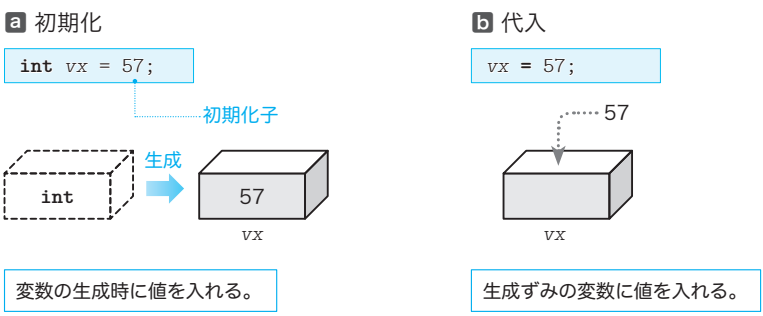


Fig.1-8 初期化と代入

演習 1–4

int 型変数の宣言に実数値の初期化子（たとえば 3.14 や 5.7 など）を与えるとどうなるだろうか。プログラムを作成して確認せよ。

1–3 読込みと表示

本節では、キーボードから整数値を読み込んで、その値を変数に格納する方法などを学習します。

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(おもしろ)くありません。以下のように、キーボードから数値を読み込んで、対話的に処理を行うようにします。

整数値を読み込んで、確認のためにその値をそのまま表示する。

プログラムを **List 1-11** に示します。

List 1-11 chap01/list0111.c

```
/*
    読み込んだ整数値をそのまま表示
*/
#include <stdio.h>

int main(void)
{
    int no;

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

    printf("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。
```

表示される値は、打ち込んだ数値に応じて変化する。

いろいろな数値を打ち込んでみよう !!

/* 整数値を読み込む */

printf とは異なって & が必要 !!

scanf 関数：読込みを行う関数

Fig.1-9 に示すように、キーボードから数値などを読み込む際に用いるのが ***scanf*** 関数です（一般に、***scanf*** はスキャンエフと発音します）。

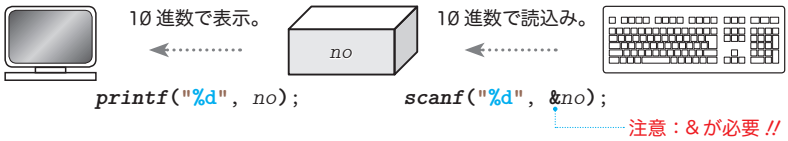
ここでの変換指定 **"%d"** は、***printf*** の場合と同様に、10 進数の指定です。したがって、

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。

と依頼することになります。なお、以下の点が要注意です。

重要 ***printf*** 関数による表示とは異なり、***scanf*** 関数による読込みでは、実引数として与える変数名の前に **&** を付ける必要がある。

▶ **&** の意味は第 10 章で学習します。また、**int** 型が格納できる数値には限りがありますので、極端に大きな数値や小さな負の数値を読み込むことはできません（第 7 章で学習します）。



● **Fig.1-9** printf 関数による表示と scanf 関数による読込み

さて、プログラムでは、まず「整数を入力してください：」と表示して、整数値の入力を促します。***scanf*** 関数による読込みが完了すると、『あなたは ** と入力しましたね。』と表示します（変数 *no* に読み込んだ値が ** の部分に表示されます）。

▶ これ以降の解説では、以下のように「 」と『 』を使い分けます。
「ＡＢＣ」と表示 … 画面に ABC と表示します。
『ＡＢＣ』と表示 … 画面に ABC と表示した後に改行します（改行文字を出力します）。

乗算を行う

読み込んだ整数値をそのまま表示するのではなく、5 倍した値を表示するようにプログラムを書きかえましょう。**List 1-12** に示すのが、そのプログラムです。

List 1-12 chap01/list0112.c

```
/*
    読み込んだ整数値の5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);                          /* 整数値を読み込む */

    printf("その数の5倍は%dです。\n", 5 * no);

    return 0;
}
```

実行例

```
整数を入力してください：357↵
その数の5倍は1785です。
```

本プログラムで初めて利用したアステリスク * は、乗算（掛け算）の記号です。もちろん、プログラム中の 5 * *no* を、*no* * 5 に変更しても、同じ結果が得られます。

演習 1–5

右に示すように、読み込んだ整数値に 12 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↵
57に12を加えると69です。
```

演習 1–6

右に示すように、読み込んだ整数値から 6 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↵
57から6を減じると51です。
```

puts 関数：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くことにします。

二つの整数値を読み込んで、その和を表示する。

このプログラムを **List 1-13** に示します。

List 1-13 chap01/list0113.c

```
/*
    読み込んだ二つの整数値の和（加算結果）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。
```

▶ 網かけ部のように、一つの行中に複数の文を置くことが可能です（逆に、一つの文が複数の行にまたがっても構いません）。プログラム表記の詳細は p.104 で学習します。

本プログラムで初めて利用したのが ***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発音します）。

その ***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、さらに最後に改行文字を出力します。すなわち、**Fig.1-10** に示すように、***puts***("…") は、***printf***("…\n") とほぼ同じ働きをします。

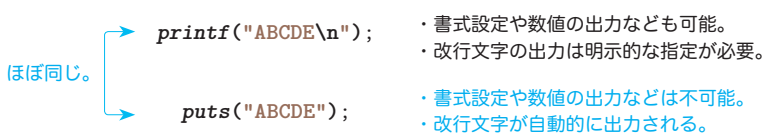


Fig.1-10 printf 関数と puts 関数

最後に改行文字の出力があって、かつ、書式化の必要がない場合は、***printf*** 関数ではなくて ***puts*** 関数を使うのがおすすめです。

▶ ***puts*** 関数に与える実引数は一つだけです。また、記号文字 % の表示方法が、***printf*** 関数とは異なります（p.23 で学習します）。

このプログラムを少しだけ書きかえたのが **List 1-14** です。読み込んだ二つの整数値の加算結果を変数 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、左ページのプログラムと同じです。

List 1-14 chap01/list0114.c

```
/*
    読み込んだ二つの整数値の和（加算結果）を変数に格納して表示
*/
#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。
```

本プログラムでは、加算した値を表示しているだけですから、変数 wa を導入するメリットはほとんどありません。しかし、加算した値をもとにして、さらに次の計算を行うようなプログラムでは、変数を導入するメリットが生まれます。

演習 1–7

『天』『地』『人』と表示するプログラムを作成せよ。表示には ***printf*** 関数ではなく ***puts*** 関数を利用すること。

```
天
地
人
```

演習 1–8

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 1–9

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。
```

まとめ

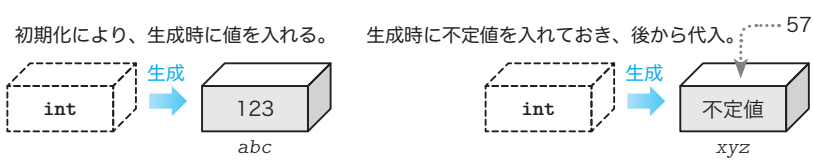
- ソースプログラムは、文字の並びとして作成する。そのままでは実行できないので、コンパイル（翻訳）やリンクを行って、実行可能な実行プログラムに変換する。
- ソースプログラム中の /* から */ までは、注釈（コメント）である。注釈は、複数行にまたがれる。作成者自身を含めた読み手に伝えるべき適切なことがらを簡潔に記入するとよい。
- 右に示すソースプロクラムの水色部分は、“おまじない”として丸暗記しておく。
- stdio.h の綴りを studio.h と間違えないようにする。
- 文の末尾には、原則としてセミコロン；が必要である。

```
#include <stdio.h>

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

    return 0;
}
```

- プログラムを実行すると、{ と } のあいだの文が順次実行される。
- 改行文字を表す拡張表記は **\n** で、警報文字（通常はビープ音）を表す拡張表記は **\a** である。
環境によっては、逆斜線記号 \ の代わりに、円記号 ¥ を使わなければならない。
- 文字の並びを表すのが、一連の文字を二重引用符 " で囲んだ "ABC" や "こんにちは。" などの文字列リテラルである。
- 数値などのデータを自由に出し入れできる変数は、“型”から作られた実体である。変数を使うには、型と名前を与える宣言が必要である。**int** 型は、整数を表す型である。
- 変数は生成されたときに不定値が入れられる。そのため、変数を宣言する際は、特に不要でない限り、初期化子を与えて初期化を行うべきである。
- 変数に値を入れる“初期化”と“代入”の違いは、以下のとおりである。
  初期化：変数を生成するときに値を入れること。
  代　入：生成ずみの変数に値を入れること。

型名 変数名 初期化子

```
int abc = 123;      /* 初期化（変数の生成時に値を入れる）*/
int xyz;            /* 不定値（ゴミの値）で初期化 */
xyz = 57;           /* 代入（生成ずみの変数に値を入れる）*/
```



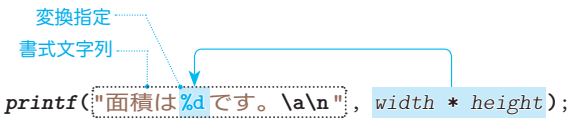
- 複数の変数を一度に宣言する場合は、以下のように変数名をコンマで区切る。

```
int a, b;
```

- 関数呼出しは、処理の依頼であり、その際に必要な補助的な指示は、( ) の中に実引数としてコンマ , で区切って与える。

- 画面への表示を行う関数としては、***printf*** 関数と ***puts*** 関数とがある。

- ***printf*** 関数に与える最初の実引数は、書式文字列である。書式文字列は、続く実引数の書式を指定するための変換指定を含むことができる。書式文字列中の変換指定以外の文字は、基本的にそのまま表示される。
  変換指定 %d は、続く実引数を 10 進数で表示するための指定である。



- ***puts*** 関数は、与えられた文字の並びを表示した上で改行文字を出力する。

- キーボードから数値などを読み込んで変数に格納する関数は ***scanf*** 関数である。実引数として与える読込み先の変数名の前には & を付ける。
  変換指定 %d は、10 進数で読み込むための指定である。

- 加算を行う記号は + で、減算を行う記号は - で、乗算を行う記号は * である。

ソースプログラムを保存するソースファイルの拡張子は .c とする。

chap01/summary.c

```
/*
    長方形の面積を求める
*/

#include <stdio.h>

int main(void)
{
    int width;      /* 長方形の横幅 */
    int height;     /* 長方形の高さ */

    puts("長方形の面積を求めます。");

    printf("横幅：");
    scanf("%d", &width);

    printf("高さ：");
    scanf("%d", &height);

    /* 表示 */
    printf("面積は%dです。\a\n", width * height);

    return 0;
}
```

実行例

```
長方形の面積を求めます。
横幅：7
高さ：5
面積は35です。
```

変数の宣言。

表示後に改行される。

表示後に改行されない。

整数値を 10 進数で読み込む。

& を忘れないようにする。

整数値を 10 進数で表示する。

乗算を行う。

\a は警報で、\n は改行を表す拡張表記。

各文が順次実行される。