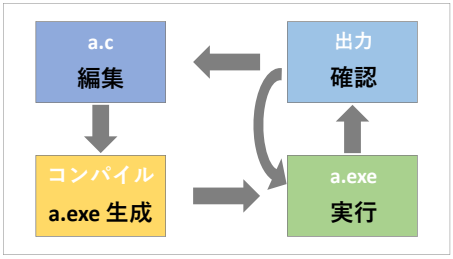
第 9 章

プログラムを作る

本章では、簡単なシェルの操作と、初めての C 言語によるプログラムを作ります。 また、最後に簡単なゲームを頑張って入力してみましょう。

プログラミングをする上で、重要なものとして**開発環境** (development environment) があります。作業の流れは、大まかに以下の図のようになります。



これらの作業を効率よく行える「統合開発環境」もあって、利便性は高いのですが、まずはこれらの作業を別々に実行することで、それぞれの役割を理解しましょう。

キーワード

- ソースコード，コンパイル
- スタンダード・アイ・オー
- main() 関数，ブロック，コメント
- エスケープシーケンス，改行文字

9.1 シェルの操作

ターミナル（例えば Cygwin Terminal）のウィンドウを開いて、次のようなコマンドを入力してみましょう。 **黄色地** の文字はキーボードから入力することを示します。

Windows や Mac では、ファイルをグループ分けした入れ物を**フォルダ** (folder) と呼びますが、元々は Unix 系で**ディレクトリ** (directory) と呼んでいるものです。コマンドにも directory のキーワードが入っています。

`ls` : (list) ファイル名の一覧を表示します。
- `ls` Enter は現在のフォルダのファイル名を表示します。
- `ls␣-l`（エル） Enter はファイルの作成日などの情報も表示します。

`pwd` : (print working directory) `pwd` Enter は現在のフォルダを表示します。

`cd` : (change directory) フォルダを移動します。
- `cd␣hoge` Enter のように、フォルダ名を指定すると、そのフォルダに移動します。1 階層下に移動することになります。
- `cd␣..` Enter は 1 階層上のフォルダに移動します。
- `cd` Enter はホームディレクトリ[*1]へ戻ります。
- `cd␣` を入力してから、ファイルマネージャのフォルダアイコンを Cygwin Terminal のウィンドウにドラッグすると**フルパス名が入力**されるので、続けて Enter を押すだけで、一度に目的のフォルダに移動できます。

`mkdir` : (make directory) フォルダを作ります。
- `mkdir␣hoge` Enter は hoge というフォルダを作ります。

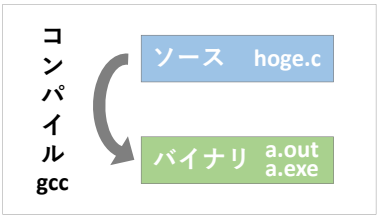
`history` : (history) `history` Enter は最近実行したコマンドを表示します。

`cat` : (concatenate) ファイルの内容を表示します。
- `cat␣hoge.txt` Enter は hoge.txt というファイルの内容を表示します。

このように、人間が対話的に操作して、プログラムを起動するためのプログラムを**シェル** (shell) あるいは**コマンドインタプリタ** (command interpreter) といいます。シェルにはいくつもの種類があって、Cygwin Terminal の場合は bash というプログラムが動いています。

ファイルマネージャ（Windows の Explorer や Mac の Finder）は、シェルの GUI 版です。コマンド版（bash）との相互作用も理解しておいてください。片方で作ったファイルが、もう片方でも出現することを確かめておきましょう。

[*1] シェルを最初に起動したときにいるフォルダのことです。環境変数の HOME でカスタマイズできます。



9.2　C 言語プログラムの実行まで

C 言語では、人間の入力したプログラムを直接実行するのではなく、実行形式に翻訳してから実行するという、2 段階の構成になっています[*2]。

この翻訳方式を採用する言語では、元になるプログラムを**ソースコード** (source code)、翻訳作業を**コンパイル** (compile)、コンパイルするプログラムを**コンパイラ** (compiler) と呼びます。（本書で想定するコンパイラは☞**??**ページ）

ソースコードを保存したファイルは**ソースファイル** (source file) といいます。C 言語のソースファイルは、拡張子を `".c"` にします。生成される**実行ファイル** (executable file) のファイル名は、Unix 系では `"a.out"`、Cygwin では `"a.exe"`[*3] が既定値です。Windows の Visual C++ の cl などでは、ソースファイルの拡張子を `".exe"` に変えたものです。

呼び名を短くして、ソースコードを**ソース** (source)、実行ファイルを**バイナリ** (binary)[*4]ということもあります。

1. ソースコード（*.c）を編集
2. コンパイル（a.out や *.exe を生成）
3. 実行
4. 実行結果の確認

思い通りの動作をするプログラムになるまで、この 4 つの作業を何度も繰り返します。

> ＿＿＿＿ Windows のファイルマネージャの初期設定では拡張子が表示されないので、この作業に不都合です。設定を変更して、拡張子を表示させましょう。＿＿＿＿

[*2] プログラムを翻訳せずに実行するインタプリタ（逐次解釈）方式に比べて、時間をかけて実行ファイルを生成できるので、実行効率がよいと謳われています。
[*3] Windows では、拡張子の `".exe"`, `".com"`, `".bat"` が実行ファイルの目印になっています。
[*4] バイナリの直訳は「2 進数の」ですが、ここでは「CPU が直接理解できる機械語の」という意味です。

9.3 初めてのプログラム

では、ソースコード 9.1 をテキストエディタで入力してみましょう。ファイル名は "hello.c" としてください。

ソースコード 9.1　初めてのプログラム (hello.c)

```
1 #include <stdio.h>Enter
2 Enter
3 int main(void) {Enter
4 Tab printf("Hello, World!\n");Enter
5 Tab return 0;Enter
6 }Enter
```

インデント

9.3.1 ソースコードの入力

ソースコードに使える文字は限られていて、アルファベットと数字や記号、正確には**ASCII 文字**（☞**??**節）です。アルファベットの大文字と小文字は、区別します。バックスラッシュ（ \ ）とエンターキーの注意は**??**ページを参照してください。

空白（**スペース** (space)）の空き具合は、最初はサンプルを真似してください。

- スペースも ASCII 文字にします。いわゆる「全角スペース」はエラーになります。エディタによっては、スペース文字を視覚的に表示する機能があります。
- 行末には、スペース文字をたくさん並べて次の行まで送るのではなく、Enter キー（↵）で**改行文字** (line feed character) を入力します。改行文字もエディタによっては特殊文字として表示できます。
- 行頭にある隙間を**字下げ**とか**インデント** (indent) といいます。インデントするには Tab キーを押します。1 文字で広い幅の開く**タブ文字** (tab character) が入力されるか、複数のスペース文字に展開されるか、どちらになっても構いませんが、Tab キー 1 度だけで適当な（4 文字ほどの）空き具合にならなければ、テキストエディタの設定を見直してください。拡張子によって動作が切り替わるので、最初に「名前をつけて保存」する必要もあります。

逆に、文字の大きさや色、書体は自由に選んでもらって構いません。たいていのテキストエディタには、予約語（特別な役目のキーワード）に色をつける機能があります。等幅フォントにしておくと、文字数が数えやすくてよさそうに思います[*5]。

[*5] 英語圏の中には、ソースコード上でも「等幅よりもプロポーショナルフォントのほうが読みやすい」という人もいます。縦に揃えるという概念がないのかもしれません。

9.3.2 コンパイル・実行

ソースコードが入力できたら、コンパイルしましょう。GNU Compiler Collection (GCC) を例にすると、C 言語コンパイラのコマンドは "gcc" です[*6]。シェル上で gcc␣hello.c Enter と入力します。

文法エラー (syntax error) がなければ、実行ファイルが生成されます。 ls Enter (Windows のコマンドプロンプトなら dir Enter) で確かめます。

次は実行です。生成された実行ファイルを指定します。

Unix 系・Mac	./a.out Enter
Cygwin	./a.exe Enter あるいは ./a Enter
Windows	hello.exe Enter あるいは hello Enter

いずれでも、「Hello, World!」の文字が表示されたら成功です。

```
[Cygwinのgccの場合]
$ gcc hello.c

$ ls
a.exe     hello.c

$ ./a.exe     (Unix系なら ./a.out)
Hello, World!
```

```
[WindowsのVisual C++の場合]
C:\Users\taro>cl hello.c

C:\Users\taro>dir /w
hello.c  hello.exe

C:\Users\taro>hello.exe
Hello, World!
```

- 先頭に "./" が必要なのは、シェルの制限（セキュリティ上の安全策）のためです。
- Windows や Cygwin では、拡張子部分の ".exe" は省略可能です。

コラム： コンパイルメッセージの読み方 (1)

コンパイル時に文法エラーが出た場合は、**最初のエラー**が重要です。メッセージがスクロールして画面の外に流れていっても、戻して先頭部分を読みましょう。

ソースファイル名と**行番号**が書いてありますので、まず場所を特定しましょう。

「**エラー: expected ';' before 'int'**」'int' の前にセミコロン (;) が抜けていることを指摘していますが、実際には**直前の行末**を指していることがあります。

「**エラー: プログラム内に逸脱した '\357' があります**」このような文字コードの入ったメッセージが 3 回連続したら、マルチバイト文字の混入を疑いましょう。

☞12 ページに続く

[*6] Mac の Xcode にも gcc コマンドがありますが、実体は Clang になっています。明示的に "clang" コマンドを使っても構いません。Windows の Visual C++ なら "cl" コマンドです。

9.3.3　記号の読み

C 言語ですぐに必要な記号の読みをまとめました。必ずすべて読めるようになってください。もっと多くの記号は**??**節にまとめました。

`.`	ピリオド，ドット	`*`	アスタリスク
`,`	コンマ，カンマ	`/`	スラッシュ
`:`	コロン	`\`	バックスラッシュ，逆スラッシュ
`;`	セミコロン	`'`	シングルコーテーション，一重引用符
`&`	アンパサンド，アンド記号	`"`	ダブルコーテーション，二重引用符
`|`	バーティカルライン，縦棒，パイプ	`_`	アンダーバー，アンダースコア，下線

9.3.4　プログラムの説明

- `#include` は外部ファイルを読み込む命令です。役目は**??**項で説明します。
- `<stdio.h>` はファイル名で、標準入出力（standard input / output）を意味します。発音は「**スタンダード・アイ・オー**」で、「スタジオ」とは意味も綴りも違います。
- 最初に `main()` 関数が実行されると決まっています。
- `printf()` は画面に文字を表示する関数です。
- 1 つの**文** (statement) の終わりはセミコロン (`;`) です。
- { から } までは**ブロック** (block) と呼ばれる、プログラム上の重要な要素です。開始と終了の対応も重要で、どちらが抜けてもコンパイルエラーが大量発生します。
- "〜" で囲まれた部分は、**文字列定数** (string constant) あるいは**文字列リテラル** (string literal) といいます。そのままメッセージとして扱われて、記号や漢字を含めてもエラーになりません[*7]。

9.3.5　スペース・改行・タブ

C 言語のソース上の**スペース**文字は、何文字並べても、あるいは**改行文字**や**タブ文字**に置き換えても、文法上は同じ意味になるところが多くあります。つまり人間の都合で**スペーシング** (spacing)（空白をどれだけ空けるか）を調節できるので、プログラムが読みやすくなるよう工夫しましょう。行頭の**インデント**は、これから特に重要になります。

コンマ（,）とセミコロン（;）の後ろには、スペースを 1 個入れるのが習慣です。（行末では不要です。）これは英文タイプライターの規則とも合致します。

[*7] 文字列リテラルにマルチバイト文字が入ってもエラーにはなりませんし、開発環境と実行環境が首尾一貫していれば、何とか化けずに表示されるでしょう。ただし Shift_JIS はいくつかの文字がおかしくなるので、コンパイラ側に対策が必要です。なお Java では、ソースコードの文字エンコードを指定できて、実行環境との整合性を言語が保証します。

TODO: 文字→文字列

9.4　コメント

プログラムを作る上で、メモを残したいこともあるでしょう。作成日とか、参考にしたURL とか、プログラムの動作自体を書き留めると有益です。

ソースコード上の**コメント** (comment) とは、プログラムとしては何の動作もしない領域のことです。コメントには、このようなメモを書き残せます。C 言語では 2 つの形式があります。

複数行コメント /* から */ まで（例：通常 `/* コメント */` 通常）
1 行コメント　// から行末まで（例：通常 `// コメント`）

注意すべき点として、複数行コメントは、さらに複数行コメントで囲おうとしても、入れ子にはできません。コメント開始の /* が何回あっても、1 回の */ でコメント終了です。（例：通常 `/* コメント /* コメント */` 通常）

1 行コメントは、C99 で正式に言語規格に取り入れられました。C++ で先に採用されていたこともあり、コンパイラの独自拡張でサポートされていた期間も長くありました。

本書のサンプルコード中でも、説明のためにコメントを活用します。コンパイルエラーになる部分は、コメントにして実行できないことを表します。

コラム： コメント

コメントはプログラム上の重要な機能です。メモを残すことはもちろんですが、一時的に動作をやめてみるのにも、コメントが役立ちます。

ちなみに、プログラムの一部をコメントにすることを、日本語でも「コメントアウト」(comment out) と、英語表現をそのまま使う人もいます。(comment out は動作です。出来上がったコメント領域を「コメントアウト」と呼ぶのは珍妙です。) 逆にコメントをやめることは、英語では uncomment といいますが、日本語では「アンコメント」という人は少ないです。

9.5　複数行表示

では次に、表示するメッセージを 2 行に増やしてみましょう。それには、画面上で**改行**の起こる仕組みを理解しておく必要があります。

9.5.1　コンソールと改行

コンピュータの黎明期には、本当に文字だけのやりとりをする機器（キーボードとモニタ）があって、**コンソール** (console) と呼ばれていました。プログラムの出力する文字は、コンソールのモニタ上に表示されていました。今ではこの動作を、ソフトウェアで模倣しています。**仮想ターミナル** (virtual terminal) とも**ターミナルエミュレータ** (terminal emulator) とも、単に**ターミナル**とも呼ばれる、ウィンドウシステム上のソフトです。

つまり、C 言語のプログラムから見ると、相変わらず**コンソール**を通じて文字のやりとりをしているのですが、我々はソフトウェアの**仮想ターミナル**上で操作しています。このため、例えばマウスの操作は C 言語プログラムへの指示になりません。

コンソールでの文字表示には癖があって、タイプライターのような動作をします。つまり 1 文字表示すると、次に表示する位置は右にずれます。画面の**右端まで到達**すると、次の行（つまり下に）進んで、表示位置が左端に戻ります。この動作を**改行**といいます。**改行文字**を表示しようとすると、何も表示されずに、**行の途中**でこの改行動作をします。

9

9.5.2　文字列リテラル中の改行文字

ダブルコーテーションで囲われた "〜" の文字列リテラルには、たいていの文字を書くことができると説明しましたが、ソースコード上の 1 行で完結する必要があります。つまり、（Enter で入力する）本物の改行文字を含めるわけにはいきません。文字列リテラルでは**エスケープシーケンス** (escape sequence) を用いて \n と表記します。（☞**??**節）

```
/* エラーになる例 */
printf("Hello, World!
"); // 途中で改行してはいけない
```

```
/* 正しい例 */
printf("Hello, World!\n");
// ここで改行が起こる↑
```

この改行文字（\n）の役目をよく理解しましょう。ソースコード上で 2 行に分かれていることと、実行してコンソールに 2 行表示されることは**無関係**で、\n を何回表示しているかが重要です。次の 2 つの例は、どちらも同じ 1 行を表示します。

```
printf("Hello, World!\n");
```

```
printf("Hello, ");
printf("World!\n");
```

タイプライターは
左上から右へと
そして下へと
タイプする

複数行表示する方法はいくつもあります[*8]。以下の例は、すべて同じ動作をします。よく使われるのは、最初の (A) と最後の (E) です。

(A) \n を 1 回含む printf() を繰り返します。

(B) 1 行で printf() を繰り返します。ソースコード上の改行は、スペース文字と同じ意味です。ソースコードが横長になって読みにくくなります。

(C) printf() は 1 回で、1 つの文字列リテラル中に改行文字を何度も登場させます。

(D) 文字列リテラルを分割します。少し驚きですが、**連続する文字列リテラルは連結**して解釈されるので、このようなことができます。

(E) 分割した文字列リテラルを複数行に振り分けます。ソースコード上のスペースと改行は同じ意味なので、このようなことができます。これが実際の表示イメージにもっとも近いでしょう。


```
/* (A) printf()を2回 */
printf("Hello, World!\n");
printf("Hello, World!\n");

/* (B) 1行でprintf()を2回 */
printf("Hello, World!\n");  printf("Hello, World!\n");

/* (C) 1行で \n を2回 */
printf("Hello, World!\nHello, World!\n");

/* (D) 文字列リテラルを2分割 */
printf("Hello, World!\n" "Hello, World!\n");

/* (E) 分割した文字列リテラルを2行に */
printf("Hello, World!\n"
       "Hello, World!\n");
```

[*8] 他の言語に目を向けると、たいていのスクリプト言語では、ヒアドキュメント（here document）という機能で、文字列を複数行にわたって羅列できます。Java では文字列を簡単に連結できます。

9.6　簡単な計算

コンピュータは計算機とも呼ばれるくらいですから、計算は得意です。簡単な計算をさせてみましょう。詳しい文法は次章以降で説明しますので、まずはソースコード 9.2 の通りに入力してください。

プログラムは、最初に main() 関数が実行されると決まっているので、**必ず main() 関数を作って**ください。実行すると「6」の数値が表示されるはずです。printf() は、%d をそのまま表示するのではなく、コンマの後の計算式の値に置き換えて表示します。

ソースコード 9.2　簡単な計算

```
1  #include <stdio.h>
2
3  int main(void) {
4      printf("%d\n", 1 + 2 + 3);
5      return 0;
6  }
```

ソースコード 9.2 の実行結果

```
6
```

次はソースコード 9.3 で変数を使ってみましょう。int というのは、その後ろの変数が整数を格納することを示します。2 つの変数 a, b を作って、その値を表示してみました。

ソースコード 9.3　変数を用いた計算

```
1  #include <stdio.h>
2
3  int main(void) {
4      int a = 1;
5      int b = a + 3;  // 1 + 3 = 4
6      printf("a = %d, b = %d\n", a, b);              // a = 1, b = 4
7
8      printf("aとbの和は %d, 差は %d\n", a+b, a-b); // 和は 5, 差は -3
9      return 0;
10 }
```

printf() に複数の %d があれば、コンマで区切られた値を順番に使います。ここでは 2 つの値があるので、表示は「a = 1」のように区別がつくよう工夫してみました。そして、和（a+b）と差（a-b）の値も計算しました。

ソースコード 9.3 の実行結果

```
a = 1, b = 4
aとbの和は 5, 差は -3
```

このプログラムを改造して、変数を増やしたり、演算の種類も変えてみてください。掛け算の×はキーボードにないので、* で代用します。割り算の÷も / で代用しますが、みなさんの想像する動作とは異なるかもしれません。

鋭意作成中

ここまでは整数の計算をしてきましたが、次は小数の計算です。ソースコード 9.4 では、行列 $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ と、その逆行列 $\frac{1}{ad-bc}\begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$ を表示しています。

ソースコード 9.4　逆行列の計算

```c
#include <stdio.h>

int main(void) {
    double a = 1.0;
    double b = 2.0;
    double c = 3.0;
    double d = 4.0;
    double t = a * d - b * c; // 行列式
    printf("%f %f -> %f %f\n", a, b,  d/t, -b/t);
    printf("%f %f    %f %f\n", c, d, -c/t,  a/t);
    return 0;
}
```

double というキーワードは、名前からは想像しにくいのですが、変数が小数を格納するという指示です。数値にも小数点をつけて 1.0 のようにします。printf() での表示には %f を使います。行列式の $ad - bc$ は何度も使うので、変数 t を作って代入しています。

ソースコード 9.4 の実行結果

```
1.000000 2.000000 -> -2.000000 1.000000
3.000000 4.000000    1.500000 -0.500000
```

これもいろいろ改造してみてください。特に $\left(\begin{smallmatrix} 1 & 2 \\ 2 & 4 \end{smallmatrix}\right)$ のように、逆行列の存在しないときに何が起こるのか、試しておきましょう。

コラム：コンパイラはどうやって作られるか

C 言語のコンパイラも、実は C 言語のプログラムとして作られているものがあります。特に GCC は、動作チェックも兼ねて、gcc のソースコードを gcc コマンドでコンパイルします。

9.7　入力と出力

プログラムに限った話ではないのですが、物事の動作というのは、1) 何かを受け取り、2) 処理をして、3) 結果を出す、というものに分かれています。例えば、料理であれば、1) 材料を準備し、2) 切ったり焼いたりして、3) 皿に盛り付けると完成、という流れでしょう。もちろん実際にはこれらは様々な順序で行われています。自動販売機を例にすれば、1) お金を受け取り、2) お金の種類から受け取った金額を変更し、3) ボタンを光らせます。そして、1) ボタンを押してもらい、2) お釣りを計算して、3) 品物とお釣りを出す、という様々な作業が入ります。

このように、動作を記述するには、処理だけでなく、受け取ることと、結果を出すことが必要です。プログラムでは、これらをそれぞれ「入力」と「出力」といいます。

コラム： コンパイル時のエラーと警告

コンパイル時に文法エラーが出ると、実行ファイルは生成されないので、どうしてもソースコードを修正する必要があります。

警告 (warning) の場合は、実行ファイルは生成されるので、とりあえずの実行は可能ですが、潜在的な問題が隠れている可能性が高いです。正常に実行できたとしても、簡単に修正できるので、警告もなくすように心がけましょう。

コラム： コンパイルメッセージの読み方 (2)

☞5 ページより続く

メッセージには専門用語の英単語や、中途半端な翻訳が混じっていてわかりにくいですが、英単語は辞書を引くなどして徐々に覚えましょう。おかしな訳語でも、メッセージをそのまま検索エンジンで調べると、ズバリの原因を解説したページに行き着くこともあります。

「**警告: 関数 'prinft' の暗黙的な宣言です**」関数名を間違えると、文法エラーではないので、このようにメッセージがわかりにくくなります。(☞**??**項)

「**'WinMain' に対する定義されていない参照です**」`main` 関数の綴りを間違えると、(特に Cygwin では、このように) さらに不可解なエラーになります。

「**Device or resource busy**」文法上の間違いではなく、実行ファイルの生成に失敗しています。(☞**??**ページの頻出ミス)

9.8　簡単なゲーム

本章の最後として、簡単なゲームを作ってみます。といっても、今回はまだ C 言語について何も説明していないので、単に頑張って打ち、エラーがないようにするだけです。これまでの二つの例から分かるとおり、こういったソースコードを作る際には、打ち間違い、空白の有無などに気をつけないといけません。**TODO:** scanf() やめて atoi() にするか

ソースコード 9.5　丁半プログラム（丁か半かを 0 か 1 で指定する）

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4
5  int main(void) {
6      srand(time(NULL));
7      int i = rand() % 2;
8      int d = -1;
9      printf("0か1か => ");
10     scanf("%d", &d);
11     if (d != 0 && d != 1) {
12         printf("0か1を入れてください\n");
13     } else if (i == d) {
14         printf("当たり！\n");
15     } else {
16         printf("外れ！\n");
17     }
18     return 0;
19 }
```

では実行してみましょう。先ほどと同じようにコンパイルして、実行します。全体の動作例は右のようになるでしょう。キーボードから入力した文字は***赤色の斜体***にしています。

10 回くらい適当に 0 か 1 を入れれば 1 回くらいは当たりが出ると思います[*9]。また、0 か 1 以外を入れた場合、0 か 1 を入れるようにメッセージを出します。

本当に 0 か 1 以外だとメッセージを出すのでしょうか？いくつ試してみましたか？どんなものを試してみましたか？

丁半プログラムの実行例（最初の二回の出力結果はこうなるとは限りません）

```
$ gcc -Wall 01-chohan.c

$ ./a
0か1か => 1
当たり！

$ ./a
0か1か => 1
外れ！

$ ./a
0か1か => a
0か1を入れてください
```

[*9] おおよそ正解が 1/2 だと思うと 10 回連続で外す可能性は $1/2^{10} \fallingdotseq 0.1\%$ ですから、まあ大丈夫でしょう。

実は、このプログラムには一つ欠点があります。試しに、0a と入れてみてください。0 でも 1 でもないはずですが、これは 0 を入れたかのように動きます。また、01 や 00 のように、あまり一般的でない書き方をするものも問題なく動きます。05 はだめといわれます。

さて、皆さんはこれを大丈夫と思いますか？それともまずいと思いますか？

実際のところ、これがよいのか悪いのかは、簡単には決定できません。ある側面では、0 や 1 に相当するものが入力されているので 00 や 01 は問題ないと考えるかもしれませんし、0a は最初にあるのが 0 だから大丈夫とも考えられます。もちろん逆も考えられて、0 や 1 を入力してほしいのだから 00 や 01 は違うし、0a にいたってはまったく違う文字が混じっているのだからまずい、ともいえます。このように、プログラムの入力のよしあしはプログラムをどのような環境で使うのか、つまりどのような入力を想定しているのかによっても変わります。そのため、この問題についてはこの本では取り扱うことができません。代わりに、なぜこうなるのか、を皆さんが理解できるようにすることをこの本では重視します。

コラム：擬似乱数

ソースコード 9.5 の 7 行目の「`rand() % 2`」が 0 か 1 かどちらかの擬似乱数を生成しています。詳細は**??**節で説明します。

本物の乱数なら、次の値の予測がまったくできなくて、二度と再現できません。ところがプログラムで生成する乱数は、（わかりにくい）規則で作り出しているので、再現可能です。この点で、本物ではない「擬似」の乱数というわけです。プログラムの開発（間違い探し）には都合のよい性質でもあります。

9.9 練習問題

1. **[初めてのプログラム（☞9.3.3 項、9.3.4 項）]**
   (i) `:` `;` `'` `"` `_` の各記号の読み方を述べよ。
   (ii) `#include <stdio.h>` の「stdio」の読み方（あるいは意味）を述べよ。「スタジオ」ではない。
2. **[変数・式（☞9.6 節）]**
   10 ページのソースコード 9.3 を参考に、整数 y, m に関する、次の式の値を表示してみよ。（式の最後の「`% 7`」は 7 で割った余りを求めるので、0 から 6 の値になる。）

   ```
   (y + y/4 - y/100 + y/400 + 13*(m+1)/5) % 7
   ```

   そして、y 年 m 月 1 日（$3 \leq m \leq 12$）の曜日との関係で、気づいたことを述べよ。

索引