

魔法の Cプログラミング演習書

—入門から実践まで—

博士(理学) 倉光 君郎 著

コロナ社

まえがき

本書は、これ一冊解けば、プログラミング入門から実践的なプログラミングまでひととおり習得できる魔法の演習書を目指して書きました。

本書は、まず教える立場から、徹底的に楽できることを目標としています。「はい」と一冊渡して、あとは Web 上にプログラミング情報はあふれているのだから自ら調べながら勉強を進めてほしい、願わくは本当に実用的なプログラムが書けるようになってほしい、という願望を実現できるように編集してあります。そのため、問題を解く上で検索すべき用語を **キーワード** のように枠で囲って、どういうところに興味をもち関心を広げていくべきか示してあります。どんどん調べて、プログラミングの深淵^{しんえん}に迫ってもらいたいと思います。

学ぶ者の立場からすると、丁寧に体系だって教えてくれる教科書がほしいなと思うかもしれませんが、そのような教科書では、プログラミングの本質を身に付けるのは難しいです。プログラミングは、習った知識を活用する能力だけでは不十分で、未知の問題にアプローチする「問題解決能力」が必要だからです。ちょっと難しい問題にも挑戦することで、問題解決の姿勢を身に付けてください。本演習書は、理解して進めれば、Web で検索しても、友達に聞いても構いません。立ち止まらずに、問題を解決し、前に進むことが大切です。

本書の執筆にあたり、つぎのような点に配慮しながら、演習問題を集めてみました。

- 基礎から実践力まで、無理なくステップアップできる
- (問題数を厳選し)一題一題、新しい知見が得られる内容
- プログラミングへの興味が高まるように幅広いテーマ

本書の問題は、著者の勤務校である横浜国立大学の「プログラミング入門」や「プログラミング演習」で出題してきた過去問をベースに出題しております。解説は、学生たちが間違いやすかった点を踏まえながら、講義の語り口で解説してみました。コーディングスタイルは、著者のオープンソース開発の経験を生かし、読みやすくバグの少ないコードを書く習慣が身に付くように心掛けました。また、先に解説を読んでもあまり実害がないように、中盤以降はできるだけソースコード全文の掲載は避けてあります。

もしソースコード全文を参考にしたい場合は、著者の GitHub レポジトリを探してください。

<https://github.com/kkuramitsu/book/>

この一冊を仕上げて、プログラミングの楽しさを学んでもらいたいと思います。

本書の執筆にあたっては多くの方にお世話になりました。中でも、特に菅谷みどり氏、そして学生目線で草稿を読んでくれた千田忠賢氏、大川貴一氏に感謝します。またコロナ社には企画段階からたいへんにお世話になりました。本書で利用しているソフトウェアや技術的な内容は日ごろの研究活動のたまものです。研究室のメンバーに感謝します。最後に、こうして本を書いていただけるのも妻雅子の協力があってこそです。ありがとう。

2016年12月

倉光 君郎

問題の難易度

問題の難易度を表すレベルとして、各問題には が付いております。難易度の参考にしながら、 を集める要領で問題を解いていってください。

- 【 】 言語文法に関する基礎問題
- 【 】 初心者レベル
- 【 】 初心者卒業レベル
- 【 】 情報系学科で標準的に期待されるレベル
- 【 】 挑戦者向け。解けなくても心配しなくてもよい問題

まずは目安として、 を 50 個以上集めることを目指してください。「プログラミング入門」は完了です。 を 100 個以上集めたら、どこに行っても恥ずかしくない基礎力が身に付いているでしょう。

目 次

1. 最初の一步

1.1	Hello, world	1
1.2	表示と書式	3
1.3	変数と代入	5
1.4	演算子	7
1.5	算術ライブラリ	9
1.6	関数定義	10
1.7	抽象化	12
1.8	条件分岐	13
1.9	多値分岐	15
1.10	ループ	17
1.11	ループと再帰	20
(コラム)	C99とコーディングスタイル	22
1.12	デバッグ	23
1.13	アサーション	25
1.14	まとめ	27

2. ウォーミングアップ

2.1	最大公約数	28
2.2	再利用	31
2.3	インクルードファイル	32
2.4	精度と誤差	33
2.5	2の補数表現	36
2.6	モンテカル口法	37
2.7	数当てゲーム	39

2.8 立 方 根	41
2.9 ゴール指向で考える	43
2.10 科学的手法	45
2.11 採用試験	47

3. 配 列

3.1 循環する数列	50
3.2 統計関数	53
3.3 バブルソート	55
3.4 順 列	57
3.5 サ イ コ ロ	58
3.6 シーザー暗号	60
3.7 エラトステネスのふるい	63
3.8 ハノイの塔	65
3.9 ライフゲーム	68
3.10 メ モ 化	72
3.11 FizzBuzz 問題	74

4. ポ イ ン タ

4.1 メモリとアドレス	78
4.2 参 照 渡 し	80
4.3 void ポインタ	83
4.4 ポインタと配列	85
4.5 ヒ ー プ	87
4.6 メモリ領域	90
4.7 可変長配列	93
4.8 構 造 体	95
4.9 ライブラリとポインタ	97
4.10 ポインタによる連結リスト	99
4.11 メモリリーク	101

4.12	スタックオーバーフロー	103
4.13	セグメンテーション違反の原因	104
4.14	関数ポインタ	108
4.15	ま と め	110

5. C を 超 え る

5.1	マクロと言語拡張	111
5.2	不 変 性	114
5.3	脆 弱 性	116
5.4	文字列の連結	117
5.5	共用体とキャスト	118
5.6	部 分 型	121
5.7	型 検 査	123
5.8	オブジェクト指向への道	125
5.9	ガベージコレクション	128
5.10	例 外 処 理	131
5.11	ま と め	134

6. システムプログラミング

6.1	コマンド引数	135
6.2	コマンド実行	138
6.3	ファイル読み込み	139
6.4	CSV ファイルの出力	141
6.5	実行時のエラー処理	144
6.6	カラフルなプログラム	147
6.7	クロスプラットフォーム	148
6.8	オープンソースライブラリ	151
6.9	ま と め	154

7. データ構造とアルゴリズム

7.1 キ ュ ー	155
7.2 ス タ ッ ク	157
7.3 分 割 統 治 法	158
7.4 帰 着	162
7.5 グラフの探索	163
7.6 動的計画法	166
(コラム) 競技プログラミング	168
7.7 ま と め	169

8. 仕上げの問題

8.1 ベンチマーク	170
8.2 アルゴリズムの選択	173
8.3 ワードカウント	175
8.4 仕様変更	180
8.5 10 パズル	182
8.6 完全情報ゲーム	184
8.7 オセロ AI の対戦	186
8.8 制限付きの正規表現	188
8.9 自由課題	191
(コラム) よいコードとよいソフトウェア	192
索 引	193

1

最初の一步

C Programming

この章は、プログラミング初学者向けの基本文法を確認しながらのスタートアップです。最初のうちは、他人の書いたコードを参考にする機会も多いと思います。プログラムが動いたら満足ではなく、好奇心をもってここを変更したらどうなるだろうと試してみてください。

1.1 Hello, world

演習室で課題を解くときは、教員があらかじめプログラミング環境を用意してくれています。しかし、演習室を一步出たら、誰かがプログラミング環境を整備してくれることはありません。ということは、自分自身で、自分のパソコンに環境設定できなければ、頑張って勉強した知識やスキルはどこで使うのでしょうか？まずは、Web の情報を頼りに是非 **C プログラミング環境設定** を試してみてください。

この問題はプログラミングを始める前段階に難しさがあるので、いきなり涙の です。

問題 1.1: (自分のパソコンにプログラミング環境を設定し) C プログラムを書いて、hello, world と表示せよ。

【ヒント】 まずは内容を気にせず、つぎのコードを一字一句正確に入力してみよう

```
#include <stdio.h>
int main() {
    printf("hello, world\n");
    return 0;
}
```

【難度 Up! ()】 コンパイラは **Clang** がオススメです。

解 説

Hello, world プログラムは、C 言語の開発者 K & R (Kernighan and Ritchie) による『プログラミング言語 C』(原題: The C Programming Language) に登場する最初の C プ

プログラムです。あまりに有名なプログラムなので、プログラミング環境（C コンパイラやエディタ）の動作確認をするテストプログラムの代名詞になっています。

ここでは、標準的な Unix 開発環境が設定できたという想定で進めます。コンパイラは Clang を想定しています。

まず、vim などのエディタを使って、ヒントのコードを入力し、hello.c というファイル名に保存してください。

Hello, world プログラムでは、あまりコードの内容を考えずに動かしてみることが重要です。ただし、内容が気になる読者のために簡単に解説を付けると：

- 関数 main() は、プログラムが起動したとき、最初に行われる関数です。最後の return 0 は、プログラムが正常に完了したことを表す状態です（この辺りは、C 言語というより、オペレーティングシステムの決まり事です）
- 関数 printf() は、書式付きで文字列表示をするライブラリ関数です。この関数を利用するため、標準入出力（standard input/output）ライブラリ stdio.h を事前に include しています

C 言語は、コンパイラ型のプログラミング言語です。コンパイル&ビルドという手続きを踏んで、実行可能なプログラムを得ることができます。

標準的な Unix の開発環境であれば、hello.c は、つぎのとおり make コマンド一発でコンパイル&ビルドできます。ターミナルのプロンプトから入力してみてください。

```
$ make hello
cc    hello.c  -o hello
```

コンパイル&ビルドに成功すると、新しく hello コマンドが作成されます。同じく、Unix ターミナルのプロンプト\$ から実行し結果が得られます。

```
$ ./hello
hello,world
```

構文エラー

もしコンパイル&ビルドに失敗すると、(それはソースコードが文法的に正しくなかった場合ですが、) エラーメッセージ (error) が表示されます。error というキーワードには注意してください。つぎは、エラーメッセージの例です。

```
$ make hello
cc    hello.c  -o hello
hello.c:3:25: error: expected ';' after expression
    printf("hello,world\n")
```

```

~
;

1 error generated.
make: *** [hello] Error 1

```

エラーメッセージは、コードの上の問題箇所を行番号、例えば `hello.c:3` のように指摘してくれます。英語でメッセージが表示されたとしても、無視しないでください。もしエラーメッセージの意味が見当も付かない場合は、`error: expected ';' after expression` のように、そのまま検索すると、ヒントが得られることも多いです（ちなみに、上のエラーは文末の `;` の打ち忘れによるものです）。エラーを修正したら、再度、コンパイル&ビルドし、エラーメッセージが出力されなくなるまで繰り返します。

もちろん、「vim や make を使うのはしんどい」という人もいます。そういう場合は、`統合開発環境 IDE` を用いて演習を行っても構いません。よりプログラミングしやすい環境を整えるのも重要なプログラミング能力の一つです。

1.2 表示と書式

C プログラミングの学習において、まず初めに慣れておきたいのは、`printf` 関数と書式です。これらは、プログラムの実行結果を表示するだけでなく、プログラムが期待どおり動作しない場合、原因を探するのに使うからです。

問題 1.2: つぎは、変数 `x` に対して、キーボードから入力した整数値を読み込み、その整数値を表示するコードである。

```

#include<stdio.h>
int main() {
    int x = 0;
    printf("x=");
    scanf("%d", &x);
    printf("x=%d\n", x);
    return 0;
}

```

このコードを修正することで、変数 `x` の値を 10 進数表記と 16 進数表記で 1 行に表示するようにせよ。

【ヒント】 `printf` 書式

【例】 `x=255` なら `x=255, FF`

【難度 Up! ()】整数の代わりに、実数 0.12345 を入力したら、0.1235 1.23e-01 12.35%と表示する

解 説

C 言語の初学者には、多少、意味が理解できない記法があっても「決まり文句」だと思って、前に進まないといけなところがあります。ポインタを理解するまでは、scanf("%d", &x) は、ユーザの数値入力を変数 x に読み込むときの「決まり文句」としてください。

関数 printf() は、Hello, world プログラムでも使ったとおり、" ... " で囲まれた文字列[†] を標準出力 (stdout) に出力するライブラリ関数です。文字列の中で % で始まる書式 (format) を書くと、書式に従って数値をフォーマットして表示してくれます。なぜ書式機能があるのかといえば、整数値はコンピュータ内部ではビット列 (2 の補数表現) で表現されており、人間が読むためには 10 進数や 16 進数の表現に変換する必要があるからです。書式は、その変換の役割を担当しています。

整数の 10 進数表記は %d, 16 進数表記は %x となります。問題は、「10 進数表記と 16 進数表記で 1 行に出力する」ということなので、改行 (\n) の前に 2 種類の書式を書いて、変数 x を 2 回、それぞれの書式で表示できるようにします。

```
printf("x=%d %x\n", x, x);
```

改行 (\n) は文字列の中で特別な文字コードを表すエスケープシーケンスと呼ばれる記号です。改行以外に、ダブルクォート (\"), バッククォート (\) など、文字列中にそのまま表現できない文字を表すのに使います。書式と併せて、エスケープシーケンスも調べておきましょう。

浮動小数点数の出力

課題 は、実数の入力と出力です。ただし、C 言語では、厳密な意味で、実数を扱うことはできません。実数を近似したデータ型として、単精度浮動小数点数 float 型 と倍精度浮動小数点数 double 型 が用意されています。特に理由がないかぎり、精度の優れた double 型 を使います。double 型 の変数にユーザ入力を読むときは、scanf の書式は %d の代わりに %lf を使います。

```
double x = 0.0;
scanf("%lf", &x);
```

[†] 文字列は、文字が並んだデータで、ポインタ操作を覚えると、整数値と同じくプログラムから操作できます。

これも、標準入力から `double` 型の変数を読み込む「決まり文句」としてください(混乱しやすい話ですが、あとから述べる関数 `printf()` の書式とは別物です。`%f` ではありません)。

`double` 型の `printf()` の書式は、`%f` と `%e` の 2 種あります。それぞれ小数点表記、指数表記を意味します。また、オプションで桁数を指定することができます。例えば、つぎのようになると、それぞれ小数点以下 4 桁、2 桁の指定となります。

```
%.4f    %.2e
```

百分率表記は、専用の書式がありません。表示する前に、 $(x * 100)$ のように 100 倍して、百分率に変換します。最後の単位の `%` は、検索力を試すような意地悪な問題でしたが、`%%` という特殊な書式を使います。結構、`\%` と間違えてしまう箇所です。

さて、まだ 問題なので、ソースコード全文を掲載しておきましょう。

```
#include <stdio.h>
int main() {
    double x = 0.0;
    printf("x=");
    scanf("%lf", &x);
    printf("x=%.4f %.2e %3.2f%%\n", x, x, (100*x));
    return 0;
}
```

`printf()` の書式は、C 言語の主要な型ごとに用意されています。新しいデータ型を学んだら、書式を確認して正しく表示できるようにしましょう。

1.3 変数と代入

数学は得意だけど、どうもプログラミングは苦手という学生は少なくありません。C 言語は、数式の記法は数学を基盤にしていますが、数学とは本質的に異なる世界です。両者の違いを理解し、発想を切り替えることが必要です。

問題 1.3: $n = n + 1$ は、数式としては(矛盾しているので)成り立たないが、プログラムとしてはある正しい操作であることをプログラム実行例と共に示せ。

解 説

C 言語では、演算子 `=` は「代入 (assignment)」と呼ばれる操作をする演算子です。つぎの式は、「変数 `n` に対し、変数 `n+1` の結果を代入する」という意味です。

```
n = n + 1
```

索引

【あ】	関数ポインタ	108	再利用	31
アサーション	関数を呼ぶ	11	先入れ先出し	155
値渡し	間接参照	81	先読みミニマックス法	184
アップキャスト			サニタイジング処理	139
アドレス演算子	【き】		算術ライブラリ	9
アロー演算子	記号定数	40	参照化	81
	偽 値	8	参照カウント方式	129
	帰 着	163	参照透過性	73
【い】	基本型	121	参照渡し	81
インタフェース	キャスト演算子	34		
インデックス	キュー	155	【し】	
インデント	共用体	119	シーザー暗号	60
インライン展開			式	7
	【く】		事後条件	26
【え】	クイックソート	160, 174	事前条件	26
エスケープシーケンス	グローバル変数	79	実行時エラー	144
エラーメッセージ			車輪の再発明	10
エラトステネスのふるい	【け】		ジャンプテーブル	15
エントリ	警 告	24	主記憶装置	78
	計算機イブシロン	42	巡回セールスマン	169
【お】	計算量	162	条件コンパイル	41, 149
オープンアドレス法	契約的プログラミング	26	条件分岐	13
オープンソースライブラリ	ゲーム木	185	真 値	8
オブジェクト				
オブジェクト不変性	【こ】		【す】	
愚かなキャスト	構造体	95	スコープ	6
	構文解析	188	スタック	157
【か】	コードレビュー	23	スタックオーバーフロー	25, 103
カーネル領域	ゴール指向	44	スタックマシン	157
ガベージコレクション	コールスタック	25	スタック領域	79
カウンティングソート	コールバック関数	153	スワップ	56, 83
返り値	コマンド引数	136		
型安全性	コンストラクタ	127	【せ】	
型強制	コンパイラ型	2	正規表現マッチング	139
型検査	コンパイラ最適化	172	脆弱性	116
可変長配列			整数オーバーフロー	74
可変引数	【さ】		静的型検査	123
関数インタフェース	再帰下降構文解析法	158, 189	静的型付き言語	10
関数コール	再帰関数	21	静的メモリ確保	88
関数適用	再帰構造	17	静的領域	79
関数の返り値	最大公約数	28	セグメンテーション違反	
関数の戻り値	最適化	172		

——の連結	118	ライブラリ関数	9	連結リスト	99
文字列マッチング	170	ラッパー関数	146	連鎖法	180
戻り値	11	乱数生成	37		
モンテカルロ法	37			【ろ】	
【ゆ】		【り】		ローカル変数	6, 79
ユークリッドの互除法	28	リエントラント	91	論理積	8
有限状態オートマトン	189	リスト	99	論理値	8
		リファクタリング	30	論理和	8
【よ】				【わ】	
余算	184	【る】		ワードカウント	175
		ループ構造	17	ワンライナー	30
【ら】		【れ】			
ライフゲーム	68	例外処理	132		

【A】		【H】		Open GL	154
ASCII コード	61	hello,world	1	Open SSL	154
【B】		【I】		【P】	
BM 法	171	IDE	3	printf 書式	3
Boehm GC	131	IPA セキュアプログラミング 講座	117	printf デバッグ	7
【C】		ISO-2022-JP	141	【S】	
calloc	88	【K】		SDL	154
Clang	1	KMP 法	171	SIMD ベクトル化	87
const 修飾子	114	KR 法	171	sizeof 演算子	84
CSV	143	【L】		SJIS	141
【D】		Lex/Yacc	158	SQLite	154
DFA	189	Libcurl	152	【T】	
DFS	164	LRU	180	Thompson 構成法	191
double 型	4	【M】		try/catch	132
【E】		main	2	typedef	125
error	2	malloc	88	【U】	
EUCJIS	141	Map-Reduce	175	Unicode	141
【F】		MT	60	UTF-8	141
FIFO	155	【N】		【V】	
final 修飾子	114	Newton-Raphson 法	43	void ポインタ	83
FizzBuzz 問題	74	NFA	189	【記号と数字】	
float 型	4	NULL	89	\"	4
【G】		NULL ポインタ	89	\\	4
GC	128	【O】		1 行コーディング	30
GNU GMP	74	OpenCV	154	10 パズル	182
				2 の補数表現	4, 36