

はじめてのC言語

練習帳

アトリエ未来【著】

★1の基本計算問題から
石取りゲーム、数当てゲームや

★5のポーカー、双六まで全50問

C言語を愛する方々にお勧めの一冊です

第四版

はじめてのC言語 練習帳

[著] アトリエ未来

「技術書典 15 新刊」
令和五年十一月十一日 ver 4.0.0

■免責

本書は情報の提供のみを目的としています。

本書の内容を実行・適用・運用したことで何が起きようとも、それは実行・適用・運用した人自身の責任であり、著者や関係者はいかなる責任も負いません。

■商標

本書に登場するシステム名や製品名は、関係各社の商標または登録商標です。

また本書では、™、®、©などのマークは省略しています。

始めに

楽しいプログラミングの世界へようこそ。

情報技術「IT」に囲まれた生活を送るわたくしたち。多くの先人が築いた歴史の上に今日があります。^{こんにち}があります。

導入篇では、C言語が使われている場所や開発された方々、技術的な特徴などを紹介しています。

練習篇では、計算問題から始まり、条件分岐や繰り返し、配列や文字列の操作、棒グラフやカレンダーの表示、双六やポーカーなど楽しいゲームの作成など、様々な例を出題いたしました。楽しみながら、ご自身の腕試しとしてプログラミングに挑戦していただければと思います。

作成篇では、一例として著者のコードを示しました。テキストの補足として解説も加えてございますので、ご参考になれば幸いです。

巻末には、今後の成長へ繋げて欲しいとの願いから、参考書籍とC言語の簡易まとめを付けました。

現代の魔法、それがプログラミングです。自由自在にコンピュータを操って、幸せな未来へと大きく羽ばたいていってください。

♣ 対象読者

パソコンの操作ができる、初心者の方を想定しています。プログラミングに興味があって、なにか作ってみたい人の最初の一歩になればと願っています。

令和の御代は、小学校からプログラミングに親しむ時代。C言語を学ぶ方の練習帳として、あるいは、誰かに教えたいと思っている方が学習希望者にお渡しするテキストとして、お読み頂ければ幸いです。

♣ プログラムの作り方

C言語によるプログラムの作成は、次の手順で行います。

1. エディタでソースコードを書く。
2. コンパイラで、コンピュータが理解できる言語（機械語）にコンパイル（翻訳）する。
3. 出来上がった実行ファイルを実行する。
4. 意図通りに動作するか確認（テスト）する。
5. もし、改善点があれば1に戻ってコードを修正（デバッグ）する。
6. 意図通りに動作するものが出来上がったら完成です。

1. のエディタについては、筆者は Pulsar^{*1} を愛用しております。お好みのエディタでプログラミングを楽しんでください。

^{*1} <https://pulsar-edit.dev>

2. のコンパイラは、GCC (GNU Compiler Collection) 13.2.0 で動作確認しております。Macをお使いの方は [Homebrew^{*2}](#) から、Windowsをお使いの方は [Mingw-w64^{*3}](#) からインストールすることができます。

また、お手元のコンピュータにコンパイラをインストールするのではなく、ブラウザから実行できるサイトとして [paiza.io^{*4}](#) があります。短いプログラムをすぐに実行して結果を確かめることができますので、こちらを利用するのも良いかと思います。

ソースコードを書いて、最初から意図通り動くことは少ないでしょう。4. の動作確認（テスト）や、5. の不具合の修正（デバッグ）を繰り返しながら、少しづつ改善していくこととなります。

`;(セミコロン)` は付いているでしょうか、括弧の対応は取れているでしょうか、`if` 文での条件分岐や、`for` 文などの繰り返し条件は適切でしょうかなど、見直すと良いです。また要所要所で `printf` 文を入れ、変数の値を出力させると、プログラムがどのように動いているのか、確かめることができます。不具合の修正を行う際の助けとなります。

分かりやすいプログラムのためには、良い名前をつけることも大切です。`a` よりは `number` のほうが、何を表す変数なのか表明されており、コードを読み解く上で助けとなります。

また、プログラムを書き始める前に、どのような方針で作るのか、紙の上で大まかに手順を考えるのも良い方法です。キーボードからの入力を受け取って、こういう繰り返し処理をして、表示したら良さそうだと、見通しを立てて、それぞれの役割を果たす部品、機能を作成していくと、スムーズに開発していくことができます。

プログラムの書き方はさまざまですが、筆者の解答もその一例です。どのような部品を用意して作成していくのか思案のしどころですし、はじめから大きなプログラムを書いていくよりは、少しづつ動作確認しながら進めていくと、間違いも発見しやすく、達成感もありますので、おすすめです。

出来上がったプログラムが自分の書いた通りに動くのはとっても気持ちの良いものです。プログラミング楽しみましょう。

♣ プログラムのダウンロード

この本で作成したコードは、GitHub https://github.com/Atelier-Mirai/workbook_source_code で公開していますので、適宜ご利用下さい。

^{*2} <https://brew.sh/ja/>

^{*3} <https://www.mingw-w64.org>

^{*4} <https://paiza.io/ja/projects/new>

♣ 謝辞

Re:VIEW Starter^aを用いて、快適に執筆することができました。作者のkauplanさんに厚く御礼申し上げます。

^a <https://kauplan.org/reviewstarter/>



また、表紙絵の女の子は、千葉県松戸市在住のフリーランス SD イラストレーター 早瀬ひろむ^aさんの作品です。素敵なイラストを描いてくださいり、ありがとうございます。

^a <https://hiromu-hayase.tumblr.com>



早瀬ひろむ さん

背景の向日葵と海の絵は、がらくった^aさんの作品です。向日葵と青い空と青い海、とっても楽しくなります。

^a <https://www.ac-illust.com/main/profile.php?id=aromaru>



がらくった さん

また、信頼できる文献に触れて欲しいとの思いから、ウィキペディア等、各種文献より引用させていただいております。貢献に感謝するとともに、厚く御礼申し上げます。

♣ 著者紹介



卓越した技能を有する者として認められる国家資格「応用情報技術者」を保持。平成 30 年より「アトリエ未来^a」を創業。HTML 講座や Ruby 講座などプログラミングの個人指導や、IT パスポート講座等の資格講座の開催、ウェブサイト作成等を受注している。

趣味の将棋は、日本将棋連盟より三段の免状を允許。日本の美しい自然や豊かな精神性を宿す熊野古道を歩くことや、たくさんの花に囲まれた日々を愛している。

^a <https://atelier-mirai.net/>

【コラム】盤上の夢 百万石

8 1 枠の小宇宙。将棋には無限の夢があります。

将棋が好きなお殿様、褒美をやることとします。

「なんでも望みの品を申せ」

「それではお言葉に甘えまして、将棋盤の最初の枠目には一粒のお米を、次の枠目には倍の二粒のお米を、更にその次の枠目には更に倍の四粒のお米をというように、最後の枠目まで米粒を賜りたく存じます」



さて、何粒のお米を賜ることができるでしょうか。

- (1) 米百俵
- (2) 蔵が立つほど
- (3) 山ほど一杯

(正解はこの本を最後まで読んでね。)

目次

始めに

i

第I部 導入篇

1

第1章 C言語の紹介

3

1.1	暮らしに生きるC言語	3
1.2	C言語を創った人々	6
1.3	C言語の特徴	8
	♣ 特徴	8
	♣ 自由度	8
	♣ 処理系の簡素化	9
	♣ その他	10
	♣ コード例	11
	♣ 主な制御構造	11
	♣ 誕生	11
	♣ UNIX 環境と C 言語	11
	♣ パソコンと C 言語	11
	♣ 現在の C 言語	12
1.4	C言語の規格	13
	♣ K&R	13
1.5	関連するプログラミング言語	14
	♣ 先祖	14
	♣ 繙承・拡張・サブセット	14

第II部 練習篇

15

第2章 練習問題

17

2.1	難易度 ★☆☆☆☆	17
	♣ こんなちは世界	17
	♣ いろいろな計算	17
	♣ 時刻に応じた挨拶	17
	♣ 横棒グラフの表示	17
	♣ 日々の積み重ね	18
	♣ 時間の計算	18
	♣ 合計が一万を超えるときの数	18

♣ 掛算九九	18
♣ 干支を求める	18
♣ 配列の要素の合計	18
♣ 野菜の摂取量	18
♣ 鶴亀算	18
2.2 難易度 ★★☆☆☆	19
♣ 消費税	19
♣ 円の面積	19
♣ 黄金比	19
♣ BMI(Body Mass Index)	19
♣ さいころ	19
♣ お天気予報	19
♣ 横棒グラフの表示	20
♣ 配列の最小値	20
♣ 計算ゲーム	20
♣ 素数	20
♣ 完全数	20
♣ 二の幂乗	20
♣ かぐや姫	20
♣ 盤上百万石 八十一枒の夢	21
2.3 難易度 ★★★☆☆	22
♣ 定期預金	22
♣ 干支を求める	22
♣ 福引き	22
♣ 開年	22
♣ お天気予報	23
♣ 白銀比	23
♣ 単語の長さ	23
♣ 縦棒グラフ	23
♣ 計算ゲーム	23
♣ 配列の要素の並び替え	24
♣ 石取りゲーム	24
2.4 難易度 ★★★★☆	25
♣ 成績発表	25
♣ 世界人口	25
♣ 干支を求める	25
♣ 生きてきた日数	25
♣ カレンダー	25
♣ 漢数字	26
♣ 英単語帳アプリ	26
♣ 計算ゲーム	26

♣ 数当てゲーム	26
♣ ビットマップフォント	26
2.5 難易度 ★★★★☆	27
♣ ポーカー	27
♣ 双六ゲーム	27
♣ マージソート（併合並び替え）	27
第 III 部 作成篇	29
第 3 章 作成例	31
3.1 作成例 ★☆☆☆☆	31
♣ こんにちは 世界	31
♣ いろいろな計算	31
♣ 時刻に応じた挨拶	32
♣ 棒グラフの表示	32
♣ 日々の積み重ね	33
♣ 時間の計算	34
♣ 合計が一万を越えるときの数	35
♣ 掛算九九	36
♣ 干支を求める	36
♣ 配列の要素の合計	37
♣ 野菜の摂取量	38
♣ 鶴亀算	39
3.2 作成例 ★★☆☆☆	40
♣ 消費税	40
♣ 円の面積	40
♣ 黄金比	41
♣ BMI(Body Mass Index)	42
♣ さいころ	42
♣ お天気予報	44
♣ 横棒グラフの表示	45
♣ 配列の最小値	46
♣ 計算ゲーム	47
♣ 素数	48
♣ 完全数	50
♣ 二の幂乗	51
♣ かぐや姫	52
♣ 盤上百万石 八十一枚の夢	54
3.3 作成例 ★★★☆☆	58
♣ 定期預金	58
♣ 干支を求める	58

♣ 福引き	59
♣ 開年	60
♣ お天気予報	63
♣ 白銀比	64
♣ 単語の長さ	66
♣ 縦棒グラフ	67
♣ 計算ゲーム	68
♣ 配列の要素の並び替え	71
♣ 石取りゲーム	73
3.4 作成例 ★★★★☆	76
♣ 成績発表	76
♣ 世界人口	77
♣ 干支を求める	79
♣ 生きてきた日数	81
♣ カレンダー	84
♣ 漢数字	87
♣ 英単語帳アプリ	91
♣ 計算ゲーム	93
♣ 数当てゲーム	96
♣ ビットマップフォント	98
3.5 作成例 ★★★★★	101
♣ ポーカー	101
♣ 双六ゲーム	116
♣ マージソート（併合並び替え）	122
付録 A 珠玉の名著のご紹介	129
♣ 教養としてのコンピューターサイエンス講義	129
♣ 達人プログラマー（第2版）熟達に向けたあなたの旅	129
♣ コーディングを支える技術	130
♣ みんなのコンピューターサイエンス	130
♣ プログラマの数学	130
♣ C言語による標準アルゴリズム事典	131
♣ アルゴリズム図鑑 増補改訂版 絵で見てわかる33のアルゴリズム	131
♣ Cの絵本—C言語が好きになる9つの扉	131
♣ アルゴリズムの絵本-プログラミングが好きになる9つの扉	131
♣ 新・C言語入門-シニア編-C言語実用マスターシリーズ	132
♣ C言語ポインタ完全制覇	132
♣ 小一時間でゲームをつくる7つの定番ゲームのプログラミングを体験	132
♣ リーダブルコード より良いコードを書くためのシンプルで実践的なテクニック	132
付録 B C言語 簡易まとめ	133

♣ コメント	133
♣ データ型	133
♣ リテラル	133
♣ 文字列	134
♣ 演算子	134
♣ 制御構造	136
♣ データアクセス	136
♣ 関数宣言	136
終わりに	137

第Ⅰ部

導入篇

第 1 章

C 言語の紹介

1.1 暮らしに生きるC言語

暮らしの中の様々なところで、コンピュータが使われています。そしてコンピュータプログラムの大きな部分を占めるのが「C言語」です。OS(オペレーティングシステム)や組み込み系など様々なところでC言語は使われています。



コンピュータ (Mac / iPad / iPhone)

なんといってもコンピュータの代表です。いろいろなウェブサイトを見たり、書類作成などお仕事に活用したり、そして「プログラミング」など。iPad でお絵描きや読書、iPhone で連絡を取り合ったり、写真や音楽、ゲームなどを楽しむことも出来ます。ロケットを打ち上げ、宇宙観測や天気予報に活かしたり、都市計画から住宅設計、工場で車や船、飛行機など様々なも

のを清算したり、音楽や映画、アニメーションなど芸術の分野に至るまで、様々な分野でコンピュータは使われています。

衣服

「天衣無縫」 - 「天人や天女の着物には縫い目がないという意から、詩文などが、よけいな修飾がなく、自然でわざとらしくなく完成されていること。また、人柄が純真で素直で、まったく嫌みがないさま。物事が完全無欠であることの形容 (goo 辞書)」

衣食住は、人が生きる上で生活の基盤となる要素です。大麻、木綿、絹糸、^{いにしえ}古の昔から日本人の身を纏う衣服に用いられてきました。縦糸と横糸を編んで一枚の布にして、布から切り取って縫いあわせて一着の衣服を仕立てていきます。コンピュータの活用により編み機から直接衣服を生み出すことが出来るようになりました。縫い目がないから着心地も良く、布の無駄もないとの特徴を持ちます。夢であった「天衣無縫」が顕現した瞬間です。

炊飯器

美味しい御飯を炊き上げてくれる炊飯器。「初めちよろちよろ、中ぱっぽ、赤子泣いてもふた取るな」と、朝早くから起きて竈で御飯を炊くのはなかなか難しいものでした。キャンプで飯盒炊飯した経験をお持ちの方もいらっしゃると思いますが、火加減が難しく焦げになってしまったり芯が残ってしまったこともあるかと思います。火力の調整をしたり、毎朝御飯が炊き上がるためのタイマー機能など、小さなコンピュータ（マイコン）を組み込むことで、美味しい御飯を頂けるようになりました。

エアコン

部屋にあるエアコン。暑い夏には涼風を、寒い冬には暖風を送り、快適に過ごせるよう室温を調整しています。部屋の温度を感じるセンサー機能、設定温度を保つように計算する演算機能、実際に風を送り出す送風機能から成り立っています。

給湯器

台所や浴室の給湯器もコンピュータの産物です。昔の人のお風呂として思い浮かぶのは、五右衛門風呂でしょうか。川や井戸から水を汲んできて、薪でお湯を沸かして、湯加減を確かめてと、お風呂は贅沢品でしたので、庶民は銭湯へ通いました。今ではとても簡単に給湯器を設定すると、浴槽一杯に給湯してくれ、温かいシャワーも使うことが出来ます。

信号機

会社や学校へ行く途中に見かける信号機。これにもコンピュータが使われています。赤信号、青信号と、色を変えるのはもちろん、道路の状況に応じて、近くの信号機と連携、青信号の長さを調整することで渋滞緩和を図るなどしています。

バス

通勤通学に電車やバスを利用する方も多いでしょう。例えば「さくら高校 行」と電光掲示板で行先表示したり、現金の他、ICカードを^{かざ}すことで乗車代金を払うことが出来ます。車両自体のガソリンや電気自動車の出力を制御する際や、バス停に「停留所を発車しました」と運行状況表示するなどの用途にも使われています。写真は、茨城県境町の「自動運転バス」^{*1}で、

^{*1} 自治体初！ 境町で自動運転バスを定常運行しています (<https://www.town.ibaraki-sakai.lg.jp/page/page00>)

地域の足として活躍しています。

こうして見てきただけでも、身の回りのいろいろなところにコンピュータが使われていることが分かります。他にはどこに使われているでしょうか。家の中で、お店で、学校や会社でなど、探してみましょう。

1.2

C言語を創った人々

C言語を開発した方々を Wikipedia から引用しつつ、ご紹介いたします。



ブライアン・カーニハン

ブライアン・カーニハンは、ベル研究所に在籍していたカナダ出身の計算機科学者である。C言語や UNIX の開発者であるデニス・リッチャー、ケン・トンプソンと共に、C言語および UNIX に対する多くの研究開発結果による貢献で知られている。

デニス・リッチャーと共に著の『プログラミング言語 C』(通称: K&R)は、事実上の規格書として扱われ、現在でも古典的な教科書の一つである。

多くのプログラミング言語入門書で、最初のプログラムとして書かれる `Hello world` は、彼がベル研究所で書いた B言語のチュートリアルで初めて使われた。



デニス・リッチャー

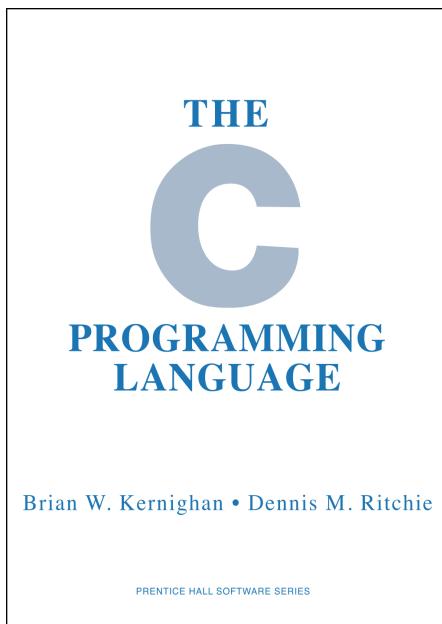
アメリカ合衆国の計算機科学者。1969年頃、同僚のケン・トンプソンと共に、ベル研究所で独自のオペレーティングシステム UNIX を作り始める。この UNIX 上で動作するアプリケーション作成の為に、トンプソンによって B言語が開発され、リッチャーがこれにデータ型と新しい文法等を追加し C言語が出来た。1973年にアセンブリ言語で書かれていた UNIX を C言語で書き換えることに成功した。C言語の開発は、リッチャーの UNIXへの最大の貢献である。

1983年、UNIX開発の功績により、ケン・トンプソンと共にチューリング賞を受賞している。今日、C言語は組込システムからスーパー・コンピュータまであらゆるプラットフォームで用いられ、彼の業績は偉大である。



ケン・トンプソン

ケン・トンプソンは、アメリカ合衆国の計算機科学者。長年ベル研究所に勤め、オリジナルの Unix を開発した。また C 言語の前身である B 言語を開発した。2006 年から Google で勤務しており、Go を共同開発した。他の主な業績として、正規表現、テキストエディタ QED と ed、UTF-8 コードの定義に加え、チェスの終盤定跡データベースやチェスマシン Belle の開発などコンピュータチェスへの貢献がある。1983 年に彼の長年の同僚であるデニス・リッチャーと共にチューリング賞を受賞した。「信用を信頼することについての考察」は、トンプソンハックとして知られる、セキュリティに関する重要な研究成果である。



▲図 1.1: プログラミング言語 C

1.3 C言語の特徴

C言語 シーゲンゴ は、1972年にAT&Tベル研究所のデニス・リッチー氏が主体となって開発したプログラミング言語です。

C言語の特徴などを Wikipedia から引用しつつ、ご紹介いたします。

♣ 特徴

- 汎用性が高い。プログラムの自由度や、目的に応じた拡張が容易であるため、パソコンソフトからゲームの作成、機械制御やシステム管理など、あらゆる分野に適応している。
- 対応する機器の範囲が広い。パソコンはもちろん、自動車や家電の組込み用マイコンからスーパーコンピュータまで、C言語を使用できるハードウェアは多様である。多目的性と、対応機器の多彩さのため、「コンピュータを使ってやること」は大抵、C言語で対応可能である。
- 商用・非商用を問わず、採用ソフトウェア分野が広い。作成や使用のための補助的なソフトウェアが豊富である。
- 開発時期が古いことから、文法に機械語の影響が強く、仕様自体は単純ではあるが明快ではなく難解である。この欠点を改良するためのうちに開発された後発言語に比較し、プログラマが記述しなければならないことが多く、低水準言語のように面倒で習得しにくい側面を持つ。
- アマチュアからプロ技術者まで、プログラマ人口が多く、プログラマのコミュニティが充実している。C言語は使用者の多さから、正負の両面含め、プログラミング文化に大きな影響を及ぼしている。
- C言語は手続き型言語である。コンパイラ言語とOSを念頭に設計している。ハードウェアをある程度抽象化しつつも、必要に応じて機械語やアセンブラーのコードと同じことを実現できるようなコンピュータ寄りの言語仕様になっている。低水準な記述ができる高級言語とも、高級言語の顔をした低級言語とも言うことがある。
- Cコンパイラは、移植の容易性、自由度、実行速度、コンパイル速度などを追求した。代わりにコンパイル後のコードの安全性を犠牲にしている。また、詳細を規格で規定せず処理系に委ねている部分が多く、C言語で書かれたソフトウェアでは処理系依存のコードが氾濫する原因となった。セキュリティー上の脆弱性や潜在的バグによる想定外の動作、コンパイラによる最適化の難しさといった問題を抱えており、最適化するとコンパイル速度が遅くなるなどの欠点が生じることがある。
- UNIXおよびCコンパイラの移植性を高めるために開発してきた経緯から、オペレーティングシステムカーネルおよびコンパイラ向けの低水準記述ができる。

♣ 自由度

- 文の区切りを終端記号セミコロン「;」で表し、改行文字にも空白にもトークンの区切りとしての意味しか持たせない「フリーフォーマット」という形式を採用している。中括弧{ }によ

るブロック構造およびスコープをサポートする。

- ALGOL の思想を受け継いで **構造化プログラミング** に対応している。手順を入れ子構造で示して見通しの良い記述をすることができる。
- モジュール化がファイルを単位として可能。モジュール内だけで有効な名前を使うことが出来るスコープを持っている。
- プログラムを戻り値つきのサブルーチンに分離できる。C言語ではこれを関数と呼び、関数内のプログラムコードでは、独立した変数が使用できる。これにより、データの流れがブロックごとに完結し、デバッグが容易になり、また関数の再帰呼び出しも可能となる。また、多人数での共同開発の際にも変数名の衝突が回避しやすくなる。
- C言語では、main関数と、標準ライブラリの printf, scanf 関数（およびその類型の関数）は、引数が可変という特殊な性格の関数である。K&Rでは、この特殊な関数 main と printf を使った例を最初に示している。
- システム記述言語として開発されたため、高級言語であるがアセンブラー的な低水準の操作ができる。ポインタ演算、ビットごとの論理演算、シフト演算などの機能を持ち、ハードウェアに密着した処理を効率よく記述できる。これはオペレーティングシステムやデバイスドライバなどを記述する上では便利であるが、注意深く利用しないと発見しにくいバグの原因となる。ライブラリ関数は、C言語規格が規定している関数と、OSが規定している関数との間の整合性、棲み分けなどが流動的である。
- 組み込みの整数型および浮動小数点数型のほか、構造体、共用体、列挙体（列挙型）によるユーザー定義のデータ型や列挙定数をサポートする。構造体および共用体はビットフィールドをサポートする。
- 多くの処理系がインラインアセンブラーを搭載しているほか、アセンブラーで出力したオブジェクトとのリンクが容易になっている。これにより速度が要求される部分だけをアセンブリ言語で記述するということが容易に行えることが多い。

♣ 処理系の簡素化

プログラムの内容によっては、以下に対して脆弱性対策を施しても実行速度の低下が無視できる程度であることも多く、欠点とみなされることも少なくない。

- 配列参照時の自動的な添字のチェックをしないこれを要因とする代表的なバグが、固定長のバッファ領域をはみだしてデータの書き込みが行われてしまう「バッファオーバーフロー」である。範囲外のアクセスは、書き込みだけでなく読み取りの場合も未定義動作を引き起こす。標準ライブラリにはバッファオーバーフローや範囲外アクセスを考慮していない関数があり、かつ多用されがちなため、しばしば脆弱性の原因となる。また、Cではプログラムにより明示的に制御（動的メモリ確保）することで可変長配列の実現を可能にしているが、確保した領域の範囲外にアクセスしても自動的な伸長は行なわれない。後継言語では、標準ライブラリまた

は組み込み型により可変長配列をサポートしていたり、範囲外アクセス時には例外（実行時エラー）を送出するなどして安全性を優先していたりすることが多い。

- 文字列を格納するための特別な型が存在しない文字列には `char` 型の配列を利用する。言語仕様上に特別な扱いはないが、ヌル文字 ('\0') を終端とする文字列表現を使い、その操作をする標準ライブラリ関数がある。これは実質的にメモリ領域のポインタアクセスそのもので、固定長バッファに対して、それより長い可変長の文字列を書き込んでしまうことがあり、バッファオーバーランの元凶の1つとなっている。後継言語では文字列処理を特に強化している場合が多く、標準ライブラリあるいは言語仕様による組み込みの文字列型を提供している。
- 自動変数の自動的な初期化をしない自動変数（静的でないローカル変数）は変数の中でも最も頻繁に用いられる。初期化されていない変数を参照した場合、値は不定となるが、不定な値へのアクセスは未定義の動作であるので、コンパイラ最適化の過程で想定しない形に改変することもある。変数宣言・初期化の仕様による制限から、変数宣言の時点では初期化せず後で代入することで初期化に代えることが日常的で、誤って不定の値の変数を読み出すバグを作り込みやすい。なお自動変数の自動とは変数の領域の確保と解放が自動であるという意味であり、自動的に初期化されるという意味ではない。後継言語では、明示的な初期化が記述されていない変数は、不定値ではなくその変数の型の既定値（ゼロあるいはゼロ相当の値）で初期化される仕様になっていることが多い。

♣ その他

- 文字の大文字・小文字を区別する。
- 入出力を含めほとんどの機能が、C言語自身で書かれたライブラリによって提供される。このことは、C言語の機種依存性が低く、入出力関係ライブラリをのぞいた部分は移植性（ポータビリティ）が高いことを意味する。さまざまな機種があるUNIXの世界でC言語が普及した理由のひとつである。
- プログラムの実行に必要とするハードウェア資源が、アセンブラーよりは多いが他の高級言語よりも少なくてすむため、現在さまざまな電化製品などの組み込みシステムでも使用されている。
- 組込み向けの場合は、プログラミング言語として、アセンブラー以外では、CとC++しか用意していないことがある。その場合、他のプログラミング言語は、CやC++で書かれた処理系が存在すれば、コンパイルすることにより利用可能となることもあるが、メモリ制約などで動作しないことがある。
- ANSI/ISOにより規格が標準化された後は言語仕様の変化が小さく安定していること、C言語のプログラマ人口やコード資産が多いこと、C++やObjective-CからC言語関数を直接利用できること、また必要に応じて他のプログラミング言語からC言語関数を呼び出すためのバインディングを記述することが容易であることなどから、APIの外部仕様としてC言語の関数インターフェイスが選ばれることが多い。例えばOpenGLやOpenCLのようなオープン規格は第一級言語としてC言語を採用している。

♣ コード例

▼ Hello world プログラム

```
#include <stdio.h>

int main(int argc, char* argv[]){
    printf("Hello, world!\n");
    return 0;
}
```

C 言語は `main` 関数から実行され、`printf` 関数は変数や書式化された文字列などが表示できる比較的高機能な出力関数である。コード中の「\n」は改行を表している。

♣ 主な制御構造

- `while` 文
- `do-while` 文
- `for` 文
- `if` 文
- `switch` 文
- 関数
- `return` 文

♣ 誕生

C 言語は、AT&T ベル研究所のケン・トンプソンが開発した B 言語の改良として誕生した。

1973 年、トンプソンと UNIX の開発を行っていたデニス・リッチャーは B 言語を改良し、実行可能な機械語を直接生成する C 言語のコンパイラを開発した。UNIX は大部分を C 言語によって書き換え、C 言語のコンパイラ自体も移植性の高い実装の Portable C Compiler に置き換わったこともあり、UNIX 上のプログラムはその後に C 言語を広く利用するようになった。

♣ UNIX 環境と C 言語

アセンブラーとの親和性が高いために、ハードウェアに密着したコーディングがやりやすかったこと、言語仕様が小さいためコンパイラの開発が楽だったこと、小さな資源で動く実行プログラムを作りやすかったこと、UNIX 環境での実績があり、後述の K&R といった解説文書が存在していたことなど、さまざまな要因から C 言語は業務開発や情報処理研究での利用者を増やしていく。特にメーカー間でオペレーティングシステムや CPU などのアーキテクチャが違う UNIX 環境では再移植の必要性がしばしば生じて、プログラムを C 言語で書いてソースレベル互換を確保することが標準となった。

♣ パソコンと C 言語

1980 年代に普及し始めたパーソナルコンピュータは当初、8 ビット CPU で ROM-BASIC を搭

載していたものも多く、BASIC が普及していたが、80年代後半以降、16ビット CPU を採用しメモリも増えた（ROM-BASIC 非搭載の）パソコンが主流になりだすと、2万円前後の安価なコンピュータが存在したこともあり、ユーザーが急増した。8ビットや 8086 系のパソコンへの移植は、ポインタなどに制限や拡張を加えることで解決していた。

♣ 現在の C 言語

1990年代中盤以降は、最初に学ぶプログラミング言語としても主流となった。GUI 環境の普及とオブジェクト指向の普及により Java、Objective-C、C++、PHP、Visual Basic、などの言語の利用者も増加したため、広く利用されるプログラミング言語の数は増加傾向にある。現在でも Java, C#, C++ など C 言語の後続言語を含めて、C 言語は比較的移植性に優れた言語であり、業務用開発やフリーソフトウェア開発、C++ などの実装が困難な組み込みなどの小規模のシステムで、幅広く利用されている。

1.4

C 言語の規格

♣ K&R

リッチャーとカーニハンの共著である「**The C Programming Language**」1978年を出版。その後標準ができるまで実質的なC言語の標準として参照。C言語は発展可能な言語で、この本の記述も発展の可能性のある部分は厳密な記述をしておらず、曖昧な部分が存在していた。C言語が普及するとともに、互換性のない処理系が数多く誕生した。これはプログラミング言語でしばしば起こる現象であり、C言語固有の現象ではない。

その後、C89/C90, C99, C11, C17と仕様が整備された。

1.5 関連するプログラミング言語

♣ 先祖

ALGOL

ヨーロッパ生まれのアルゴリズム記述言語。Pascal や C 言語などに影響を与えたとされる。

BCPL

MULTICS で作成された高級言語。

B 言語

初期の UNIX で作成されたインタプリタ方式の高級言語。BCPL を元に作られ、C の原型となつた。

♣ 繙承・拡張・サブセット

C++

C 言語を拡張してオブジェクト指向化したもの。Simula の影響を強く受けている。当初は C 言語のスーパー・セットだったが、現在は細かい部分において非互換仕様が増えている。

Objective-C

C 言語を拡張してオブジェクト指向化したもの。C 言語に Smalltalk のオブジェクトシステムを取り付けたような設計で、互換性は保たれている。C 言語からの拡張部分が C++ と干渉しないため、C++ と混在した記述が可能。

Java

C++ よりも言語文法レベルでオブジェクト指向を重視した言語。バッファオーバーランなどの危険性が高いポインタといったローレベルな要素を言語文法から排除している。仮想マシン (Java VM, JVM) 上で動作する。

C#

マイクロソフトが.NET Framework 向けに開発した言語。文法は C 言語および C++ に近い書式を持ち、Java と似ている部分も存在するが、機能的には Delphi がベースとなっている。

Rust

C 言語および C++ に代わるシステムプログラミング言語を目指している言語。言語レベルでの RAII の強制による自動メモリ管理機構を持ち、ガベージコレクション無しでも手動のメモリ管理が不要であり、実行性能は C/C++ と同等である。

Unified Parallel C

並列計算向けに C99 を拡張して作られた言語。

D 言語

C 言語をベースとし ABI 互換を保つつもり、テンプレートによるジェネリックプログラミングやオブジェクト指向プログラミング、関数型プログラミングなどをサポートするマルチパラダイムプログラミング言語である。

第 II 部

練習篇

第 2 章

練習問題

2.1 難易度 ★☆☆☆☆

画面への表示、キーボードからの入力、数値計算、条件判断、繰り返し、一次元配列など、プログラミングの基礎に関する出題です。

♣ こんにちは 世界

古き良き伝統に従って、「こんにちは 世界」とご挨拶するプログラムを作ってみましょう。

♣ いろいろな計算

- $3 + 8 \times 3 - 1$ はいくつでしょうか？
- $18 \div 3 - 1$ はいくつでしょうか？
- $18 \div (3 - 1)$ はいくつでしょうか？
- 31を7で割った余りはいくつでしょうか？

♣ 時刻に応じた挨拶

今の時刻を尋ねます。

朝なら「おはようございます」
昼なら「こんにちは」
夜なら「おやすみなさい」

と挨拶するプログラムを作ってみましょう。

♣ 横棒グラフの表示

横棒グラフを表示してみましょう。

■ ■ ■ ■ ■	(5個 ■ 0個 □ があります)
■ ■ ■ ■ □	(4個 ■ 1個 □ があります)
■ ■ ■ □ □	(3個 ■ 2個 □ があります)
■ ■ □ □ □	(2個 ■ 3個 □ があります)
■ □ □ □ □	(1個 ■ 4個 □ があります)

いろいろな解き方がありますので考えてみましょう。

♣ 日々の積み重ね

毎日 1 %ずつこつこつ成長していったら、一年後には、何倍の実力になっているでしょうか？

♣ 時間の計算

123456789 秒は、何年何日何時間何分何秒でしょうか？

♣ 合計が一万を越えるときの数

$1 + 2 + 3 + \dots$ と数を足していきましょう。どの数を足したときに、合計が 10000 を越えるでしょうか？

♣ 掛算九九

掛算九九の表を出してみましょう。

♣ 千支を求める

干支を求めてみましょう。グレゴリオ暦 2000 年生まれの人の干支は何でしょうか？

♣ 配列の要素の合計

配列に 10 個の数字が入っています。合計を求めてみましょう。

♣ 野菜の摂取量

一日三食で、350 g の野菜を食べると良いと言われています。朝ごはん、昼ごはん、夜ごはんで食べた量を入力し、健康づくりの助言をするプログラムを作りましょう。

♣ 鶴亀算

鶴と亀が合わせて 8 羽（匹）います。足の数は合わせて 26 本です。鶴は何羽、亀は何匹いるでしょうか？

2.2 難易度 ★★☆☆☆

標準ライブラリを用いたり、自作の関数を作成する練習課題です。

♣ 消費税

江戸時代以降、三公七民でずっと豊かな暮らしを営んで参りました。今や相次ぐ増税で、五公五民、一揆が起きる水準です。

税抜98円のアイスクリームを3個買うと消費税はいくら徴収されるでしょうか？

♣ 円の面積

円の面積を求めてみましょう。円周率は、 $3.141592653589\ 793238462643\ 383279502884\dots$ と続きます。よく使うので定数として用意されています。

```
#include <math.h>           // 円周率 M_PI が定義されています
printf("円周率 = %f", M_PI);
```

円周率 M_PI を使って、円の面積を求めてみましょう。

♣ 黄金比

長方形から、その一辺の長さに等しい正方形を取り除いた時に残る長方形が元の長方形と相似である時、長辺と短辺の比を「黄金比」と言います。

美しい人物の彫刻や絵画などにはこの黄金比が見出されるそうです。黄金比を算出してみましょう。

♣ BMI(Body Mass Index)

太りすぎ、痩せすぎなど、体格を表す指標としてBMIがあります。

体重 [単位 kg]/(身長 [単位 m] の二乗) で計算します。

身長と体重を入力すると、太りすぎか痩せすぎか分かるプログラムを作りましょう。基準は22が標準体重、25以上の場合は肥満、18.5未満である場合低体重です。

♣ さいころ

さいころを転がして、「偶数です」「奇数です」と表示するプログラムを作ってみましょう。

♣ お天気予報

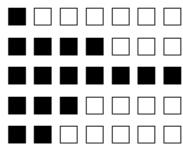
傘を持っていった方がよいかどうか、アドバイスしてくれるプログラムを作りましょう。

```
降水確率30%なら 「傘はいらないです」
降水確率70%なら 「持って行った方がいいかも」
```

降水確率90%なら 「絶対持って行きましょう」

♣ 横棒グラフの表示

横棒グラフを表示してみましょう。



前回とは違い、それぞれの行によって、表示する「■」の数が異なっています。どのように書けば良いでしょうか。

♣ 配列の最小値

配列に10個の数字が入っています。一番小さい数は何でしょうか？ その数は何番目の要素でしょうか？

♣ 計算ゲーム

計算ゲームです。10題 簡単な足し算（ $3 + 5 = ?$ など）が出題されるゲームを創りましょう。

♣ 素数

1と自分自身以外で割り切れない数のことを「素数」と言います。1～100までの間の素数を表示してみましょう。また1000個目の素数は何でしょうか？

♣ 完全数

完全数とは、自分自身の約数の和が自分自身になる数のことです。例を挙げます。

- 6は、1でも2でも3でも割り切れます。そして1と2と3を足すと6になります。
- 28は、1と2と4と7と14で割り切れます。1と2と4と7と14を足すと28になります。

28の次の完全数は、いくつでしょうか？ 10000までには、いくつ完全数があるでしょうか？

♣ 二の幂乗

二の幂乗を出してみましょう。2の10乗はいくつでしょうか？ 2の20乗はいくつでしょうか？

♣ かぐや姫

月へと帰って行ったかぐや姫の物語はとても有名です。一枚の紙を二枚に折ると厚さは二倍に、更に折ると厚さは四倍になります。何回折ると、月まで届くでしょうか。

♣ 盤上百万石 八十一枡の夢

将棋が好きなお殿様、褒美をやることとします。

「なんでも望みの品を申せ」

「それではお言葉に甘えまして、
将棋盤の最初の枡目には一粒のお米を、
次の枡目には倍の二粒のお米を、
更にその次の枡目には更に倍の四粒のお米をというように、
最後の枡目まで米粒を賜りたく存じます」

さて、何粒のお米を賜ることができるでしょうか。

それは、何俵のお米で、何石になるでしょうか。

いくつ蔵を立てれば賜ったお米を納めることができるでしょうか。

非常に多い様を称して、「山ほどの」と形容しますが、富士山何座分でしょうか。

それだけのお米を作るためには、何町歩の田んぼが必要でしょうか。

2.3

難易度 ★★★☆☆

C言語は「文字列型」を持たず、「文字」の「配列」として文字列を扱うことから、C言語は文字列操作が苦手な言語です。

また、他の高級言語と異なり、ハードウェアに近い部分を扱える点がC言語の特徴です。「ポインタ」は、メモリを意識しつつ配列や文字列操作などとも結びついています。

ポインタや文字列操作が求められる出題です。

♣ 定期預金

年利3%の定期預金に預けました。2倍になるのは何年後でしょうか？年利6%なら、年利9%なら、年利12%なら、2倍になるのは何年後でしょうか？

書式指定した文字列を返す関数として、`sprintf`文が用意されています。
使ってみましょう。

♣ 干支を求める

干支を求めてみましょう。グレゴリオ暦2000年生まれの人の干支は何でしょうか？「文字列の配列」を使って、解いてみましょう。

♣ 福引き

福引きアプリを創りましょう。

お客様>福引をひかせてください。
店員>はい、どうぞ。
福引賞品：一等賞 世界一周の旅
: 二等賞 温泉一泊二日
: 三等賞 お好み焼き食べ放題
: 残念賞 ティッシュペーパー

10回に1回は、世界一周の旅が当たるようにしてみましょう。各賞の当籤確率を変えるにはどのようにしたら良いでしょうか。

♣ 閏年

平年か閏年かを求めるプログラムを作ってみましょう。

グレゴリオ暦年が4で割り切れる年は閏年です。
但し100で割り切れる年は閏年ではありません。
しかしながら400で割り切れる年は閏年です。

コマンドラインからの引数を受け取って、グレゴリオ暦年とするように作成してみましょう。

♣ お天気予報

傘を持っていった方がよいかどうか、アドバイスしてくれるプログラムを作りましょう。

降水確率30%なら 「傘はいらないです」
降水確率70%なら 「持って行った方がいいかも」
降水確率90%なら 「絶対持って行きましょう」

コマンドラインからの引数を受け取って、降水確率とするように作成してみましょう。

♣ 白銀比

名刺などには美しく見えると「黄金比」が良く用いられますが、書籍など印刷に良く用いられる「白銀比」もあります。

【白銀比】

長方形を二つ折りにした時にできる長方形が、元の長方形と相似である時の長辺と短辺の比

A0 用紙の面積を 1 m^2 平方メートル) として、それを二つ折りにしたもののが A1, それをさらに二つ折りにしたもののが A2 と、A0, A1, A2, ... A10 まで規格されています。それぞれの紙の大きさを求めてみましょう。(ミリメートル単位で求め、端数を切り捨てます。)

♣ 単語の長さ

英単語の長さを求めるソフトです。入力された英単語の長さを求めてみましょう。

I → 1文字
love → 4文字
you → 3文字

♣ 縦棒グラフ

縦棒グラフを出してみましょう。

□ □ ■ □ □ (ヒント)
□ ■ ■ □ □ 画面に表示するときは、
□ ■ ■ ■ □ 左から右、上から下が基本です。
□ ■ ■ ■ ■ 二次元配列を使って、
■ ■ ■ ■ ■ 表示する順番を工夫しましょう。

♣ 計算ゲーム

計算ゲームを創ってみましょう。二桁や三桁、四桁の数を出題したり、足し算だけではなく、引き算や掛け算、割り算も出題されるようにしてみましょう。

♣ 配列の要素の並び替え

配列に10個の数字が入っています。大きい順から小さい順に並び替えてみましょう。

すでに配列の一番小さい数を探し出すプログラムを作っています。配列の中から、一番小さい数を見つけて、一番後ろに挿入、次に小さい数を見つけて、後ろから二番目に挿入とすると、大きい順に並びそうです。

選択ソートと呼ばれるアルゴリズムで実装してみましょう。

♣ 石取りゲーム

場に二十数個の石があります。一度に一個から三個の石を取ることができます。二人で交互に石を取り合って、最後の石を取った人の負けです。プレイヤーとコンピューターで対局するゲームを作りましょう。

2.4 難易度 ★★★★☆

♣ 成績発表

10人の名前が格納された配列と、10人の点数が格納された配列があります。成績の良い順に名前と点数を表示してみましょう。

C言語には並び替えを行うための関数として、クイックソートが用意されています。こちらを使ってみましょう。

♣ 世界人口

世界の人口は80億(8000000000)人です。一年に1%ずつ人口が増えると来年には何人になっているでしょうか？100億(10000000000)人になるのは何年後でしょうか？

また、大きな数を読みやすくするために、3桁ごとにカンマで区切り、「8,000,000,000」と表示してみましょう。

♣ 干支を求める

昭和30年生まれの方の干支は何でしょうか？(S30と入力します)

明治生まれから令和生まれまで、干支が求められるようにしましょう。

♣ 生きてきた日数

1980年1月1日生まれの人は、今日までに何日生きたことになるでしょうか？(生年月日は19800101と入力されます。)

昭和20年8月10日生まれの方は、どうでしょうか？(生年月日は3200810と入力されます。)

生後30000日目を迎えるのは、何年何月何日でしょうか？

♣ カレンダー

年と月を入力すると、その月のカレンダーを表示するプログラムを作ってみましょう。

「ツェラーの公式」を用いると曜日を求めることが出来ます。(グレゴリオ暦1年1月1日は月曜日となりますので、7で割った余りを求めてことで、曜日が計算出来ます)

```
int y; // グレゴリオ暦年
int m; // 月（但し1月、2月は前年の13月、14月として計算します）
int d; // 日
int h; // 曜日 // h が 0, 1, 2, 3, 4, 5, 6 の場合、
        // それぞれ 日曜日、月曜日、火曜日、水曜日、木曜日、金曜日、土曜日
// 年月日をもとに曜日を算出する
h = (y + y/4 - y/100 + y/400 + (13*m+8)/5 + d) % 7;
```

♣ 漢数字

算用数字を漢数字にするプログラムを作ってみましょう。無量大数まで変換できるようにしてみましょう。

```
1234567890  
→  
十二億三千四百五十六万七千八百九十
```

♣ 英単語帳アプリ

英単語が表示され、訳語を三択で選びます。

```
I → わたし  
love → 愛する  
you → あなた
```

と答えられたら、満点です。10問中、何問正解できるか、楽しみましょう。

♣ 計算ゲーム

計算ゲーム第三版です。回答にかかった所要時間も表示させてみましょう。オープニングやエンディングも付けて豪華にしてみましょう。

♣ 数当てゲーム

事前に用意された3桁の数字を、ヒントをもとに当てていくゲームです。

用意された数字が 9 4 3 だとします。9 9 9と入れると、9は正解ですので、1つ正解と表示します。3 6 9と入れると、3と9は正解ですので、2つ正解と表示します。4 3 9と入れると、(順番は違いますが) 3つとも合っていますので、3つ正解と表示します。

これを繰り返すことで、事前に用意された数字、9 4 3を当てるゲームです。

♣ ビットマップフォント

その昔、コンピュータに表示される書体は、格子状の枠目を塗りつぶすことで表現されていました。そのため大きく表示すると斜めの線のぎざぎざが目立ち、小さく表示すると文字が潰れてしまうことがありました。

ビット演算の練習のため、0～9の数字のビットマップフォントを作成し、表示してみましょう。

2.5 難易度 ★★★★★

♣ ポーカー

ポーカーを創ってみましょう。カードを配る機能、いらないカードを捨てる機能、他にはどんな機能が必要でしょうか。アスキーアートでトランプの絵柄を表示すると雰囲気が出ますね。

♣ 双六ゲーム

双六ゲームを創ってみましょう。止まった升目に「3つ進む」や、「振り出しに戻る」も創ってみましょう。どこまで進んだか分かる表示機能や、オープニング・エンディングもあると楽しいですね。

♣ マージソート（併合並び替え）

「連結リスト」は要素の追加や削除が得意なデータ構造です。次の要素を指示するポインタとその値を持つ構造体を作成し、リストの追加や削除を行ってみましょう。

マージソートは、先に登場したクイックソートと並んで高速な並び替えを行うアルゴリズムとして知られています。

並び替えを行いたいデータ列をまず二つに分割します。二つに分けたデータ列のそれぞれをさらに分割し、最終的に一つの要素のみになるまで分割します。一つの要素のみになったデータをマージ（併合）し、二つの要素が順に並びます。並び替えが完了した二つのデータ同士を更に併合ことで四つのデータの並び替えが完了します。これを繰り返すと、最終的に全データ列の並び替えができることがあります。

連結リストとの相性もとても良く、プログラミングの大切な技法の一つ「再帰」も学べます。本書の締めくくりとして、マージソートを実装しましょう。

第 III 部

作成篇

第3章

作成例

練習課題の作成例です。いろいろな書き方で、プログラムを創ることが出来ます。コメントも入れてございますので、参考にしていただければ幸いです。

3.1 作成例 ★☆☆☆☆

♣ こんにちは世界

```
▼ hello_world.c
1 // 標準入出力(STandard Input Output)のためのヘッダーファイルを取り込みます。
2 // これにより、標準で用意されている入力や出力のための関数が使えるようになります。
3 #include <stdio.h> // 標準入出力関数
4
5 // C言語では、プログラムは main関数から始まります。
6 // 先頭のintは整数型(int)の値を返すことを示します。
7 // argc は プログラム自身に与えられた引数の個数です。
8 // argv は プログラム自身に与えられた引数の配列です。
9 int main(int argc, char const *argv[]) {
10     printf("こんにちは 世界\n"); // 画面に文字列を出力します。
11
12     return 0; // 0(正常終了) を返して、終了します。
13 }
```

♣ いろいろな計算

```
▼ calculation.c
1 #include <stdio.h> // 標準入出力関数
2
3 int main(int argc, char const *argv[]) {
4     // %d は整数型の書式指定子です。
5     // 計算結果を「整数型」と解釈して表示するための書式指定子です。
6     // 足し算は「+」、引き算は「-」を使いますが、
7     // 掛け算は「×」の代わりに「*」を使います。
```

```

8 printf("%d\n", 3 + 8 * 3 - 1);
9
10 // 変数に代入してから、出力することも出来ます。
11 int answer; // 変数の宣言。整数型の変数 answer を宣言しています。
12 answer = 18 / 3 - 1; // 変数への代入。
13 // 割り算は「÷」の代わりに「/」を使います。
14 printf("%d\n", answer);
15
16 int result = 18 / (3 - 1); // 直接、変数に初期値を代入することも出来ます。
17 printf("%d\n", result);
18
19 // 余りを求める演算（剰余演算子）として、「%」が用意されています。
20 printf("%d\n", 31 % 7);
21
22 return 0;
23 }
```

♣ 時刻に応じた挨拶

▼ greeting.c

```

1 #include <stdio.h> // 標準入出力関数
2
3 int main(int argc, char const *argv[]) {
4     // 変数の宣言
5     int hour; // 今何時か、格納するための変数
6
7     printf("今何時ですか？\n"); // 入力を促すために、メッセージを表示
8     scanf("%d", &hour); // 整数型の数字を、受け取る
9
10    // 時刻に応じた挨拶をする
11    if (hour < 12) { // 午前中
12        // 「\n」は「改行文字」で、改行されます。
13        printf("おはようございます\n");
14    } else if (hour < 18) { // 夕方まで
15        printf("こんにちは\n");
16    } else { // 夜なら
17        // puts関数を使うと、改行文字も含めて出力されます。
18        puts("おやすみなさい");
19    }
20
21    return 0;
22 }
```

♣ 棒グラフの表示

▼ bar_graph1.c

```

1 #include <stdio.h> // 標準入出力関数
2
3 int main(int argc, char const *argv[]) {
4     // 横棒グラフを表示します。
```

```

5 // 直接、表示することも出来ます。
6 printf("■■■■■\n");
7 printf("■■■■□\n");
8 printf("■■■□□\n");
9 printf("■■□□□\n");
10 printf("■□□□□\n");
11
12 // 5つのグラフではなく7つのグラフにしたいなど、
13 // 数が変わった時にも対応できるよう、
14 // 繰り返し構文を使います。
15 // 変数名は、i でも良いですが、
16 // 行の変数であることが分かるようにすると良いです。
17 int row;           // 行
18 int black_number; // 黒い■の数
19 int white_number; // 白い□の数
20
21 for (row = 0; row < 5; row++) {
22     black_number = 5 - row; // 黒い■の数
23     white_number = row;   // 白い□の数
24     // 初期値は設定済みなので、省略出来ます。
25     for (; black_number > 0; black_number--) {
26         printf("■");      // ■を表示します。
27     }
28     for (; white_number > 0; white_number--) {
29         printf("□");      // □を表示します。
30     }
31     printf("\n");        // 改行します。
32 }
33
34 return 0;
35 }
```

♣ 日々の積み重ね

▼ one_percent.c

```

1 #include <stdio.h> // 標準入出力関数
2
3 int main(int argc, char const *argv[]) {
4     double capability = 1.0; // 能力
5     int i;
6
7     for (i = 0; i < 365; i++) {
8         capability *= 1.01;
9         // capability = capability * 1.01; と書くことも出来ます。
10        // 自分自身に何かの計算を行い、
11        // その結果を自分自身に代入することは、よく行われるので、
12        // 代入と計算を一緒に行える、演算子がC言語には用意されています。
13    }
14
15    printf("一年後の能力は、\n");
16    printf("%.15f\n", capability); // 小数点以下15桁表示
17    printf("です。.\n");
```

```

18 printf("\n");
19 printf("C言語 double型での計算結果\n");
20 printf("37.783434332887275\n");
21 printf("\n");
22 printf("正確な値は、有効桁数732桁で、\n");
23 printf("37."
24     "783434332887158877616604796497605460271135491591002003303933893694442"
25     "952198593811935639436889138752947230257466652966950262937798745172333"
26     "015079222338624286146825416806152531443969194556942776517247940062958"
27     "202175604957806833320549618283760329920784474440748232823522848774776"
28     "663377098517634258918092249275355047751709109700563151616706856329170"
29     "679969143031119841436101987303610665032253735962900715320344772671094"
30     "746342243980747288537748044810805431513656284728377150860725544069515"
31     "704180309669461071550627216255083200959680558767329997739256425299018"
32     "230968108183790782834451122341391699026728718809670675868494941801800"
33     "148043695322254918714677072113955042157310524945401321699479843200827"
34     "1425308713028897301180251050440194336501\n");
35 printf("です。.\n");
36 // 小数の計算は誤差がつきものです。
37 // double型で計算しましたが、有効桁数 14 桁でした。
38
39 return 0;
40 }
```

♣ 時間の計算

▼time.c

```

1 #include <stdio.h> // 標準入出力関数
2
3 int main(int argc, char const *argv[]) {
4     int time = 123456789;
5     int years;
6     int days;
7     int hours;
8     int minutes;
9     int seconds;
10
11     years    = time / 31536000; // 31536000秒で割った商が時間です。
12     time    = time % 31536000; // 31536000秒で割った余りが残り時間です。
13     days    = time / 86400;    // 86400秒で割った商が時間です。
14     time    = time % 86400;    // 86400秒で割った余りが残り時間です。
15     hours   = time / 3600;    // 3600秒で割った商が時間です。
16     time    = time % 3600;    // 3600秒で割った余りが残り時間です。
17     minutes = time / 60;     // 60秒で割った商が分です。
18     seconds = time % 60;     // 60秒で割った余りが残り時間です。
19     printf("%d 年 %d 日 %d 時間 %d 分 %d 秒 です。\n",
20            years, days, hours, minutes, seconds);
21
22 // 31536000などと大きな数が出てきました。
23 // 何の数字であるか分かりやすいよう、以下のように書くと良いです。
24     time    = 123456789;
25     years   = time / (365 * 24 * 60 * 60);
```

```

26     time    = time % (365 * 24 * 60 * 60);
27     days    = time / (24 * 60 * 60);
28     time    = time % (24 * 60 * 60);
29     hours   = time / (60 * 60);
30     time    = time % (60 * 60);
31     minutes = time / (60);
32     seconds = time % (60);
33     printf("%d 年 %d 日 %d 時間 %d 分 %d 秒 です。\\n",
34             years, days, hours, minutes, seconds);
35
36 // yearsなど上の位から順に求めてきましたが、
37 // secondsなど下の位から順に求めると、
38 // 31536000などの大きな数が出現せず、扱い易いです。
39     time    = 123456789;
40     seconds = time % 60;
41     time    = time / 60;
42     minutes = time % 60;
43     time    = time / 60;
44     hours   = time % 24;
45     time    = time / 24;
46     days   = time % 365;
47     years   = time / 365;
48     printf("%d 年 %d 日 %d 時間 %d 分 %d 秒 です。\\n",
49             years, days, hours, minutes, seconds);
50
51     return 0;
52 }
```

♣ 合計が一万を越えるときの数

▼total10000.c

```

1 #include <stdio.h> // 標準入出力関数
2
3 int main(int argc, char const *argv[]) {
4     int n;
5     int total;
6
7     // while文は繰り返す回数がわからない場合に効果的です。
8     // 条件を満たしているかどうかを先に判定しているので、
9     // 前判定型反復や前判定ループと呼ばれます。
10    // 前判定では、条件によっては一度も反復処理が行われず、終了することがあります。
11    n      = 1;
12    total = 0;
13    while (total <= 10000) {
14        total = total + n;
15        n++;
16    }
17    printf("%d を足したら、10000を越えて %d になりました。\\n", --n, total);
18
19    // 反復処理を行った後に、
20    // 条件を満たしているかどうかを判定したい場合には、
21    // do while 文を用います。
```

```

22 // 後判定型反復や後判定ループと呼ばれます。
23 // 後判定では、条件によらず一度は反復処理が行われます。
24 n = 1;
25 total = 0;
26 do {
27     total = total + n;
28     n++;
29 } while (total <= 10000);
30 printf("%d を足したら、10000を越えて %d になりました。\n", --n, total);
31
32 return 0;
33 }
```

♣ 掛算九九

▼ multiplication_table.c

```

1 #include <stdio.h> // 標準入出力関数
2
3 int main(int argc, char const *argv[]) {
4     int multiplier; // 乗数 (掛ける数)
5     int multiplicand; // 被乗数 (掛けられる数)
6
7     // 掛け算九九の表を表示する
8     printf(" 一 二 三 四 五 六 七 八 九\n");
9     printf(" の の の の の の の\n");
10    printf(" 段 段 段 段 段 段 段 段\n");
11    for (multiplier = 1; multiplier <= 9; multiplier++) {
12        for (multiplicand = 1; multiplicand <= 9; multiplicand++) {
13            printf("%3d", multiplicand * multiplier);
14        }
15        printf("\n");
16    }
17
18    return 0;
19 }
```

♣ 千支を求める

▼ eto1.c

```

1 #include <stdio.h> // 標準入出力関数
2
3 int main(int argc, char const *argv[]) {
4     int year = 2000; // グレゴリオ暦年
5
6     // if 文は 条件分岐のための基本です。
7     // 条件が単純な場合には switch case 文を使うと、
8     // すっきり書くことができます。
9
10    // 12で割った余りに応じて、干支を表示します。
11    switch (year % 12) {
```

```

12 case 0:
13     printf("申年\n");
14     break;
15 case 1:
16     printf("酉年\n");
17     break;
18 case 2:
19     printf("戌年\n");
20     break;
21 case 3:
22     printf("亥年\n");
23     break;
24 case 4:
25     printf("子年\n");
26     break;
27 case 5:
28     printf("丑年\n");
29     break;
30 case 6:
31     printf("寅年\n");
32     break;
33 case 7:
34     printf("卯年\n");
35     break;
36 case 8:
37     printf("辰年\n");
38     break;
39 case 9:
40     printf("巳年\n");
41     break;
42 case 10:
43     printf("午年\n");
44     break;
45 case 11:
46     printf("未年\n");
47     break;
48 }
49
50 return 0;
51 }
```

♣ 配列の要素の合計

▼ array_sum.c

```

1 #include <stdio.h> // 標準入出力関数
2
3 int main(int argc, char const *argv[]) {
4     // 10個の数字が入った配列
5     // {} を使ってまとめて初期値を与えています。
6     // 配列は array(アレイ) と言います。
7     int array[] = {3, 14, 15, 92, 65, 35, 89, 79, 24, 58};
8     int i;           // 配列の添字(index)
```

```

9 // 合計をまず初期化して 0 にします。
10 int sum = 0;
11 // 配列の添字は 0 から始まるので、
12 // for 文 も i = 0 と、0 から始めます。
13 for (i = 0; i < 10; i++) {
14     sum += array[i];
15 }
16
17 // 結果を出力します。
18 for (i = 0; i < 9; i++) {
19     printf("%2d + ", array[i]);
20 }
21 printf("%2d = %3d です。\\n", array[9], sum);
22
23 return 0;
24
25 }
```

♣ 野菜の摂取量

▼ vegetable.c

```

1 #include <stdio.h>
2
3 int main(int argc, char const *argv[]) {
4     int vegetable; // 野菜を食べた量
5     int total = 0; // 今まで食べた量の合計
6     int ratio; // 率
7     int goal = 350; // 目標
8     int n; // 何回目の食事か？
9
10    printf("== 野菜を一日 350 g 食べましょう ==\\n");
11
12    for (n = 1; n <= 3; n++) {
13        switch (n) {
14            case 1: printf("朝ごはん"); break;
15            case 2: printf("昼ごはん"); break;
16            case 3: printf("夜ごはん"); break;
17        }
18        printf("では、何 g 食べましたか？\\n");
19
20        // キーボードから入力された食べた野菜の量を取得します。
21        scanf("%d", &vegetable);
22        while (getchar() != '\\n') {
23            ; // 何もせずに読み飛ばす
24        }
25
26        // 合計を取る
27        // total += vegetable; と自己代入演算子を使って書くこともできます。
28        total = total + vegetable;
29        // 整数同士の割り算は切り捨てられるため、
30        // 100を先に掛けることで、答が0や1になることを防ぎます。
31        ratio = 100 * total / goal;
```

```

32 // 書式指定子にも % を使うため、
33 // printfの中で、% を表示するためには、%% を記述します。
34 printf("達成率は %d %% です\n", ratio);
35 if (n <= 2 && total < goal) {
36     printf("残り %d g の野菜を食べましょう\n\n", goal - total);
37 } else {
38     printf("\n");
39     break;
40 }
41 }

43 // 結果発表
44 printf ("今日は %d g の野菜を摂りました。\n", total);
45 if (total >= goal) {
46     printf("健康な食生活です(*^_^*)\n");
47 } else {
48     printf("もう少し野菜を食べましょう(*^_^*)\n";
49 }
50 }

```

♣ 鶴亀算

▼tsurukame.c

```

1 #include <stdio.h>
2
3 int main(int argc, char const *argv[]) {
4     // 一般的には、鶴の数をt、亀の数をkとすると、
5     // t + k = 8
6     // 2t + 4k = 26
7     // という連立一次方程式になります。
8     // これを解くためのアルゴリズムとして、
9     // ガウス法などがあります。
10    // ここでは、単純に力づくで解くこととします。
11    int tsuru;
12    int kame;
13    for (tsuru = 0; tsuru <= 8; tsuru++) {
14        kame = 8 - tsuru;
15
16        if (tsuru * 2 + kame * 4 == 26) {
17            break;
18        }
19    }
20
21    printf("鶴は %d 羽、亀は %d 匹います。\n", tsuru, kame);
22 }

```

3.2 作成例 ★★☆☆☆

♣ 消費税

▼ tax.c

```

1 #include <stdio.h> // 標準入出力関数
2 #include <math.h> // 数学関数
3
4
5 // 消費税率
6 // 定数として定義しておくと、変更があった場合の修正が楽です。
7 #define TAX_RATE 0.08
8
9 int main(int argc, char const *argv[]) {
10    // 消費税を求めます。
11    int icecream = 98; // アイスクリームの値段
12    int price;         // お買い上げ額
13    double tax;        // 消費税額
14    double total;      // 合計額
15
16    price = icecream * 3;      // お買い上げ額
17    tax   = price * TAX_RATE; // 消費税額
18    total = price + tax;     // 合計額
19
20    // %-5d で左詰で5桁表示されます。
21    printf("お買い上げ額は、 %-5d 円です。\n", price);
22    printf("消費税は、       %6.2f 円です。\n", tax);
23    printf("合計額は、       %6.2f 円です。\n", total);
24    // 切り捨て floorは床の意味です。
25    printf("端数を切り捨てるとき、%6.2f 円です。\n", floor(total));
26    // 切り上げ ceilは天井の意味です。
27    printf("端数を切り上げると、%6.2f 円です。\n", ceil(total));
28
29    return 0;
30 }
```

♣ 円の面積

▼ circle_area.c

```

1 #include <stdio.h> // 標準入出力関数
2 #include <math.h> // 数学関数
3
4 int main(int argc, char const *argv[]) {
5    int radius = 10;           // 円の半径
6    double area = pow(radius, 2) * M_PI; // 円の面積
7                           // pow関数で、2乗を求めています。
8
9    printf("円周率 = %f\n", M_PI);      // 特に書式を指定しなかった場合
10   printf("円周率 = %.4f\n", M_PI);    // 小数点以下4桁表示
11   printf("円周率 = %.15f\n", M_PI);   // 小数点以下15桁表示
```

```

12 printf("半径 %d の円の面積は、%.15f です。\\n", radius, area);
13
14 return 0;
15 }
```

♣ 黄金比

▼ golden_ratio.c

```

1 /*=====
2 【黄金比】
3 長方形から、その一辺の長さに等しい正方形を取り除いた時に残る長方形が
4 元の長方形と相似である時、長辺と短辺の比を「黄金比」と言います。
5
6 -----
7 1 |       |       | 1
8  |       |       |
9 -----
10      x-1
11
12 大きな長方形は、長辺 x : 短辺 1
13 正方形を取り除いた後の小さな長方形は、長辺 1 : 短辺 x-1
14 これが相似なので、
15 x : 1 = 1 : x-1
16 が成り立ちます。
17 外項の積は内項の積に等しいので
18 x(x-1) = 1
19 x^2 - x - 1 = 0
20 解の公式を用いて
21     -b ± √b^2 - 4ac
22 x = -----
23             2
24
25     1 ± √5
26 = ----- と求められます。
27         2
28 =====*/
29
30 #include <stdio.h> // 標準入出力関数
31 #include <math.h>   // 数学関数
32
33 int main(int argc, char const *argv[]) {
34     double a = 1; // 浮動小数点数として計算したいので、double型として宣言します。
35     double b = -1;
36     double c = -1;
37
38     // pow は幕乗を求める関数です。
39     // pow(n, 0.5) で n の 0.5 乗、つまり √ が計算できます。
40     double x = (-b + pow(b*b - 4 * a * c, 0.5)) / 2;
41
42     printf("%f\\n", x); // 小数を表示する際は、%fを使います。
43
44     return 0;

```

45 }

♣ BMI(Body Mass Index)

▼ bmi.c

```

1 #include <stdio.h> // 標準入出力関数
2 #include <math.h> // 数学関数
3
4 int main(int argc, char const *argv[]) {
5     // 変数宣言
6     int height; // 身長
7     int weight; // 体重
8     double bmi; // BMI
9
10    // 入力処理
11    printf("身長(cm)を入力して下さい。\\n");
12    scanf("%d", &height); // scanf でキー入力を受け取ります。
13    while (getchar() != '\\n') // 改行キーが入力されるまで待ちます。
14        ;
15
16    printf("体重(kg)を入力して下さい。\\n");
17    scanf("%d", &weight);
18    while (getchar() != '\\n')
19        ;
20
21    // BMI算出
22    // int型のheightを、キャスト演算子(double)を使って、
23    // double型に変換しています。
24    // 優先順位を明確にするため、
25    // ((double)height) を丸括弧で囲ってから、
26    // 100 で割っている点に着目して下さい。
27    bmi = weight / pow(((double)height) / 100, 2);
28
29    // 結果表示
30    printf("BMI は %4.2f です。\\n", bmi);
31    if (bmi < 18.5) {
32        printf("痩せすぎです。\\n");
33    } else if (bmi < 25) {
34        printf("標準です。\\n");
35    } else {
36        printf("肥満です。\\n");
37    }
38
39    return 0;
40 }

```

♣ さいご

▼dice.c

```
1 #include <stdio.h> // 標準入出力関数
2 #include <stdlib.h> // rand関数
3 #include <time.h> // time関数
4
5 int main(int argc, char const *argv[]) {
6     int dice;
7
8     // 実行した時刻によって、異なった乱数となるように、
9     // 亂数の種を播きます。
10    srand(time(NULL));
11
12    // 6で割った余りに 1を加えることで、
13    // 1～6までの乱数が得られます。
14    dice = rand() % 6 + 1;
15    printf("サイコロの目は %d です。\n", dice);
16
17    // 2で割った余りが0なら、偶数、そうでないなら奇数です。
18    if (dice % 2 == 0) {
19        printf("偶数です\n");
20    } else {
21        printf("奇数です\n");
22    }
23
24    // 論理和を使って、このように書くことも出来ます。
25    if (dice == 2 || dice == 4 || dice == 6) {
26        printf("偶数です\n");
27    } else {
28        printf("奇数です\n");
29    }
30
31    // switch case 文を使って書くことも出来ます。
32    switch (dice) {
33        case 1:
34            printf("奇数です\n");
35            // このbreak文がないと、
36            // case 2: も続けて実行されますので、
37            // break文を書いています。
38            break;
39        case 2:
40            printf("偶数です\n");
41            break;
42        case 3:
43            printf("奇数です\n");
44            break;
45        case 4:
46            printf("偶数です\n");
47            break;
48        case 5:
49            printf("奇数です\n");
50            break;
51        case 6:
52            printf("偶数です\n");
```

```

53     break;
54 default:
55     printf("エラーです\n");
56     // default文のbreak;は不要ですが、
57     // 対称性の観点から付けています。
58     break;
59 }
60
61 // 意図的に、break文を書かず、
62 // case 5: まで、スルーさせて、
63 // 奇数で在る旨、表示させています。
64 switch (dice) {
65 case 1:
66 case 3:
67 case 5:
68     printf("奇数です\n");
69     break;
70 case 2:
71 case 4:
72 case 6:
73     printf("偶数です\n");
74     break;
75 }
76
77 return 0;
78 }
```

♣ お天気予報

▼ umbrella.c

```

1 #include <stdio.h> // 標準入出力関数
2 #include <stdlib.h> // rand関数
3 #include <time.h> // time関数
4
5 // 亂数はよく使うので、関数化してみました。
6 // 同じプログラムを何度も書かずに済むので、便利です。
7 // 自作したrandom_number()関数が定義されています。
8 // 自作のヘッダーファイルを読み込む際は、
9 // "(ダブルクォーテーション)"で囲みます。
10 #include "random_number.h"
11
12 int main(int argc, char const *argv[]) {
13     // 降水確率 0 - 100 % の範囲の乱数
14     // 補完機能が働くので、長い変数名でも良いですが、長すぎるかもしれません。
15     // 分かりやすいプログラムを書くために、
16     // 良い名前を付けてください。
17     // 降水確率は 10% 単位なので、0~10までの乱数を10倍しています。
18     int precipitation_probability = random_number(10) * 10;
19
20     // printf文の書式の中で、「%」を表示させるには、「%%」と記述します。
21     printf("降水確率は %d %% です\n", precipitation_probability);
22 }
```

```

23 // if else 文の条件式は、
24 // 大きい順、あるいは、小さい順にして、
25 // 順次、条件に合致させるようにするとよいです。
26 if (precipitation_probability <= 30) {
27     printf("傘はいらないです\n");
28 } else if (precipitation_probability <= 70) {
29     printf("持って行った方がいいかも\n");
30 } else {
31     printf("絶対持って行きましょう\n");
32 }
33
34 return 0;
35 }
```

♣ 横棒グラフの表示

▼ bar_graph2.c

```

1 #include <stdio.h> // 標準入出力関数
2
3 // 自作の関数を創って、解くことも出来ます。
4
5 // 関数のプロトタイプ(原型)宣言
6 // 機能：黒い■と白い□を表示する
7 // 引数：int black_number // 黒い■の数
8 //         int white_number // 白い□の数
9 // 戻値：なし
10 void black_white_box(int black_number, int white_number);
11
12 int main(int argc, char const *argv[]) {
13     // 変数名は、i でも良いですが、
14     // 行の変数であることが分かるようにすると良いです。
15     int row;           // 行
16     int black_number; // 黒い■の数
17     int white_number; // 白い□の数
18
19     // 表示させたい棒グラフの値配列
20     int values[] = {1, 4, 7, 3, 2};
21
22     // 値配列の要素数が5なので、5行繰り返せば良いです。
23     // 予め配列要素の要素数を求めておくと、より汎用的になります。
24     for (row = 0; row < 5; row++) {
25         black_number = values[row];      // 黒い■の数
26         // 値配列の最大値が7なので、7から引いています。
27         // 予め配列要素の最大値を求めておくと、より汎用的になります。
28         white_number = 7 - black_number; // 白い□の数
29
30         // 黒い■と白い□を表示する関数を呼び出し、表示を任せます。
31         black_white_box(black_number, white_number);
32     }
33
34     return 0;
35 }
```

```

36 // プロトタイプ宣言を先頭でしているので、
37 // 自作関数本体を、main関数の後に書くことが出来ます。
38 void black_white_box(int black_number, int white_number) {
39     // C言語では、while(0)が偽となり、while文が終了します。
40     // ですので、このようにも書くことが出来ます。
41     while (black_number--) {
42         printf("■"); // ■を表示します。
43     }
44     while (white_number--) {
45         printf("□"); // ■を表示します。
46     }
47     printf("\n"); // 改行します。
48 }
49 }
```

♣ 配列の最小値

▼ array_min_index.c

```

1 #include <stdio.h> // 標準入出力関数
2 #include <limits.h> // 整数の取り得る範囲
3
4 int main(int argc, char const *argv[]) {
5     // 10個の数字が入った並び替え前の配列
6     int array[] = {3, 15, 22, 81, 41, 83, 72, 0, 50, 33};
7     int min; // 最小値を格納する変数
8     int i;
9
10    // (a) 最小値を求める処理
11    // int型の最大値は 2147483647(=2^31-1)です。
12    // limit.h に INT_MAX として、定義されています。
13    min = INT_MAX;
14
15    // for文で配列の先頭から順に見ていきます。
16    for (i = 0; i < 10; i++) {
17        // もし、配列の要素が、
18        // これまでに判明している最小値よりも、小さかったなら、
19        // 新しい最小値が発見されたことになるので、
20        // 最小値を更新します。
21        if (array[i] < min) {
22            min = array[i];
23        }
24    }
25
26    // (b) min は 配列の最小値 0 となっています。
27    // 最小値0 が array の何番目の要素であったかを求めてみましょう。
28    int index = 0; // 最小値 min が何番目の要素であるか
29    for (i = 0; i < 10; i++) {
30        // 配列の要素と最小値が一致するか、順に比較していきます。
31        if (array[i] == min) {
32            // もし一致する要素が見つかったならば、
33            // 添字を取得して、繰り返しを抜けます。
34            index = i;
```

```

35     break;
36 }
37 }
38
39 // 結果表示
40 for (i = 0; i < 10; i++) {
41     printf("array[%d]: %2d\n", i, array[i]);
42 }
43 printf("最小値: %d 添字: %d\n", min, index);
44
45 // ここで、(a) の最小値を求める処理と、
46 // (b) の添字を求める処理が一緒に出来るのではと思えて来ます。
47 // 一緒に書くと、次のようにになります。
48
49 // 最小値を求めると同時に、
50 // 最小値 min が何番目の要素であるか求める処理
51 min    = INT_MAX;
52 index  = 0;
53 for (i = 0; i < 10; i++) {
54     if (min > array[i]) {
55         min = array[i]; // (a) 最小値を求める
56         index = i;      // (b) 添字を求める
57     }
58 }
59
60 // 結果表示
61 // もちろん同じ結果になります。
62 for (i = 0; i < 10; i++) {
63     printf("array[%d]: %2d\n", i, array[i]);
64 }
65 printf("最小値: %d 添字: %d\n", min, index);
66
67 return 0;
68 }

```

♣ 計算ゲーム

▼ calculation_game1.c

```

1 #include <stdio.h> // 標準入出力関数
2
3 int main(int argc, char const *argv[]) {
4
5     int operand1;          // 第一被演算子
6     int operand2;          // 第二被演算子
7     int result;            // 演算結果
8     int answer;            // 回答
9     int correct;           // 正答数
10    int n;                // 繰り返し回数
11
12    correct = 0;           // 10問中何問正解か数えるために、初期化します。
13
14    // 1回目、2回目と表示させたいので、

```

```

15 // for (n = 1; n <= 10; n++) と書いているところに着目して下さい。
16 for (n = 1; n <= 10; n++) {
17     // 出題処理
18     operand1 = random_number(10);
19     // 二つの0～9までの数を用意します。
20     // do while文で、異なる数になるまで、繰り返します。
21     do {
22         operand2 = random_number(10);
23     } while (operand2 == operand1);
24
25     result = operand1 + operand2; // 演算結果
26     printf("足し算ゲーム %d 回目", n);
27     printf("%d + %d = ?\n", operand1, operand2);
28
29     // 回答を受け取る
30     scanf("%d", &answer);
31     while (getchar() != '\n')
32         ; // キーバッファ読み飛ばす
33
34     // 正解発表と正答数のカウント
35     if (answer == result) {
36         printf("正解です。\n");
37         correct++;
38     } else {
39         printf("正解は %d です。\n", result);
40     }
41 }
42
43 // 総合結果発表
44 printf("10問中 %d問 正解です。\n", correct);
45
46 return 0;
47 }
```

♣ 素数

▼ prime_number.c

```

1 ****
2 * 1と自分自身以外で割り切れない数のことを「素数」と言います。
3 * 1～100までの間の素数を表示してみましょう。
4 * また1000個目の素数は何でしょうか？
5 ****
6 #include <stdio.h> // 標準入出力関数
7
8 #define TRUE 1
9 #define FALSE 0
10
11 // 素数判定関数
12 int is_prime(int candidate, int prime_numbers[], int n);
13
14 int main(int argc, char const *argv[]) {
15     // 素数の数 pi(x) ~= x / log(x) が知られている。
```

```

16 // なので10000までの素数の数は、たかだか2000 である。
17 // よって、要素数2000とする
18 int prime_numbers[2000] = {};
19
20 // = {} と書くことで、以下のように各要素を初期化したのと同等となる
21 // for (int i = 0; i < 2000; i++) {
22 //   prime_numbers[i] = 0;
23 // }
24
25 prime_numbers[0] = 2; // 2 は 偶数唯一の素数である。
26 n = 1; // 一つの素数が発見されているので
27 int candidate = 3; // 素数候補 3から始める
28 int i;
29
30 while (candidate < 10000) {
31   // 素数の判定は、is_prime関数に任せ、
32   // 素数であったら、配列に追加する
33   if (is_prime(candidate, prime_numbers, n)) {
34     prime_numbers[n] = candidate;
35     n++;
36     candidate += 2; // 奇数の候補のみ調べるので、+2 している
37   } else {
38     candidate += 2;
39   }
40 }
41
42 // 結果表示
43 // (せっかくなので、10000までの素数表示)
44 printf(" | 1 2 3 4 5 6 7 8 9 10\n");
45 printf("-----+\n");
46 for (int i = 0; i < n; i++) {
47   if (i % 10 == 0) {
48     printf("%5d | ", i);
49   }
50   if (prime_numbers[i] != 0) {
51     printf("%5d ", prime_numbers[i]);
52   }
53   // 10 個 ずつ 改行
54   if ((i + 1) % 10 == 0) {
55     printf("\n");
56   }
57   // 100個ずつ線を表示
58   if ((i + 1) % 100 == 0) {
59     printf("-----+\n");
60   }
61 }
62 printf("\n");
63 }
64
65 // 素数判定関数
66 int is_prime(int candidate, int prime_numbers[], int n) {
67   int i;

```

```

68 int precious_metal; // 素数でないにしても、貴金属ではある。
69
70 // 10000(=100*100) が 素数かどうかを調べるには、
71 // 100までの素数で割りきれるかどうか、確認すれば良い。
72 // ここでは簡単化のために、今までに判明している全ての素数で割り切れるか確認する。
73 for (i = 0; i < n; i++) {
74     precious_metal = prime_numbers[i]; // 割り切れるかどうか
75     if (candidate % precious_metal == 0) {
76         // 割り切れたなら、素数ではない。
77         return FALSE;
78     }
79 }
80 return TRUE;
81 }
```

♣ 完全数

▼perfect_number.c

```

1 ****
2 * 完全数とは次のような数のことです。
3 * 6は、1でも2でも3でも割り切れます。そして  $6 = 1 + 2 + 3$  です。
4 * 28は、1と2と4と7と14で割り切れます。
5 * 1と2と4と7と14を足すと28になります。
6 * 28の次の完全数は、いくつでしょうか？
7 ****
8 #include <stdio.h> // 標準入出力関数
9
10 // 定数定義
11 #define TRUE 1
12 #define FALSE 0
13
14 // 完全数判定関数
15 int is_perfect(int candidate);
16
17 int main(int argc, char const *argv[]) {
18     // 6 = 2 * 3
19     // 28 =  $2^2 \times 7$  である
20     // 割り切れるかどうかを調べ、
21     // 割り切った数の合計が、もとの数と一致するか調べればよい。
22     int perfect_numbers[4] = {6, 28};
23     int n = 2; // 2つの完全数が発見されているので
24
25     int candidate = 28 + 2; // 奇数の完全数は知られていないため、
26                         // 28の次の偶数を候補とする
27
28     // 最初の四つの完全数が発見されるまで繰り返す
29     while (1) {
30         // 完全数の判定は、is_perfect関数で行う
31         if (is_perfect(candidate)) {
32             // 完全数が発見されたら追加する
33             perfect_numbers[n] = candidate;
34             n++;
```

```

35     if (n == 4) {
36         break;
37     } else {
38         candidate += 2;
39     }
40     } else {
41         candidate += 2;
42     }
43 }
44
45 // 結果表示
46 for (int i = 0; i < n; i++) {
47     printf("%d 番目の完全数は %5d です。\\n", i + 1, perfect_numbers[i]);
48 }
49 }
50
51 // 完全数判定関数
52 int is_perfect(int candidate) {
53     int sum = 0; // 総和
54     int i;
55
56     for (i = 1; i < candidate; i++) {
57         if (candidate % i == 0) {
58             sum += i;
59         }
60     }
61     if (sum == candidate) {
62         return TRUE;
63     } else {
64         return FALSE;
65     }
66 }
```

♣ 二の幂乗

▼ power.c

```

1 #include <stdio.h> // 標準入出力関数
2 #include <math.h> // 数学関数
3
4 int main(int argc, char const *argv[]) {
5     int n;
6     long power = 1; // 幂乗（べきじょう）は、英語でpowerと言います。
7
8     // 2 の幂乗を求めます。
9     for (n = 1; n <= 20; n++) {
10        power *= 2;
11        printf("2 の %2d 乗は %8ld です。\\n", n, power);
12    }
13
14 // 幂乗を求める関数も用意されています。
15 for (n = 1; n <= 20; n++) {
16     power = pow(2, n);
```

```

17     printf("2 の %2d 乗は %8ld です。\\n", n, power);
18 }
19
20 // 16進数で出力する際は、%x を指定します。
21 // ここでは、power が long 型なので %lx を指定します。
22 // 10進数と16進数との対応も参考までに出力してみます。
23 printf("乗数 16進数          10進数\\n");
24 for (n = 1; n <= 16; n++) {
25     power = pow(2, n);
26     printf("%2d %11lx %13ld\\n", n, power, power);
27 }
28 for (n = 20; n <= 40; n += 10) {
29     power = pow(2, n);
30     printf("%2d %11lx %13ld\\n", n, power, power);
31 }
32
33 return 0;
34 }
```

♣ かぐや姫

▼ kaguya.c

```

1 /*=====
2 月までの距離は 約38万4400kmです。
3
4 一枚の紙を半分に折って重ねると、その厚みは二倍になります。
5 半分に折った紙を更に折って重ねると、その厚みは元の紙の厚みの四倍になります。
6 更に折って重ねると、厚みは八倍になります。
7
8 一枚のA4用紙を何回折ったら、月まで届く厚みになるでしょうか。
9 紙の厚みは「1024枚で10cm」とします。
10 =====*/
11
12 #include <stdio.h> // 標準入出力関数
13 #include <math.h> // 数学関数
14
15 int main(int argc, char const *argv[]) {
16
17     // 10cmを何倍したら、月までの距離38万4400kmになるでしょうか。
18     // 10cmを10倍すると100cm=1mです。
19     // 1m を1000倍すると1kmです。
20     // 1km を 384400倍すると、月までの距離になります。
21     // つまり、
22     // 10cmを、10 * 1000 * 38万4400倍 = 38億4400万倍すれば、月に届くことが分かります♪
23     //
24     // 厚さ10cmの紙を一回折れば、厚さ20cmと二倍に、
25     // もう一回折れば、厚さ40cmと四倍になりますから、
26     // 「2を何回掛けたら 38億4400万を超えるか」を求める問い合わせに帰着します。
27     //
28     // 2を掛け続けて38億4400万を超えるまで繰り返す、
29     // というプログラムを書くこともできますが、
```

```

30 // ここでは対数を使って解くことにします。
31 // 対数とは、大まかに何桁の数か？を表す演算です。
32 //
33 //    10          3
34 // 2    = 1024 ≈ 10
35 //
36 // ですので、
37 //
38 //    1          0.3
39 // 2    = 10
40 //
41 // であることが分かります。
42 // これを 対数(log)を使って書くと、
43 //
44 // log 2 = 0.3 と書くことができます。
45 //    10
46 //
47 // つまり、2は、10の0.3乗であるということです。
48 // すると、4は10の0.6乗であることがすぐ分かります。
49 //          0.6      9      9.6
50 // 38億4400万倍 ≈ 40億倍 = 4 * 10億倍 = 10    * 10    = 10
51 //
52 //    1          0.3
53 // 2    = 10
54 //    2          0.6
55 // 2    = 10
56 //    3          0.9
57 // 2    = 10
58 //
59 // ・・(中略)
60 //
61 //    32          9.6
62 // 2    = 10
63 // とすぐに求めることができます。
64 //
65 // 厚さ10cmの紙を32回折ると、約40億となることが分かりました。
66 //
67 //          10
68 // 一枚の紙は、10回折ると、2 = 1024枚となり、10cmとなりますので、
69 // 一枚の紙を 10回+32回 = 42回 折ると、月まで届くことが分かりました。
70
71 double log10_2        = log10(2);           // 約 0.3
72 double log10_38oku    = log10(3844000000); // 約 9.6
73 double times         = log10_38oku / log10_2; // 約 32
74
75 long two_32_power = (long)pow((double)2, (double)32);
76 long two_42_power = (long)pow((double)2, (double)42);
77
78 printf(" %20.6f\n", log10_2);
79 printf(" %20.6f\n", log10_38oku);
80 printf(" %20.6f\n", times);
81
82 printf(" %14ld\n", two_32_power);

```

```

83 printf(" %14ld\n", two_42_power);
84
85 printf("\n === 参考 二の幂乗 === \n");
86 int n;
87 for (n = 0; n <= 42; n++) {
88     printf("%2d %14ld\n", n, (long)pow((double)2, (double)n));
89 }
90
91 return 0;
92 }
```

♣ 盤上百万石 八十一枠の夢

▼ yume.c

```

1 /*=====
2 日本人に欠かせない、お米。
3
4 一人が一年に食べるお米の量は一石と言われています。
5 加賀百万石と言われますが、
6 おおよそ百万人の人が食べていけるだけのお米が取れたのです。
7 江戸時代の人口が約三千万人で、石高も約三千万石でした。
8
9 一合 一合枠で量るお米の量です。約180mlで、重さは150gになります。
10 一升 十合です。一升瓶、一升枠などで量ります。約1.8Lです。
11 お米一粒一粒はとても小さいので、千粒纏めて重さを量ります。
12 千粒重（せんりゅうじゅう）と言い、お米の品種にもよりますが、約23gです。
13 なので、一升には約65000粒のお米が含まれていることになります。
14 一升に含まれるお米の数について、64827(虫や鮒)という覚え方もあります。
15 一斗 十升です。一斗缶に入った油などです。約18Lです。
16 一石 十斗です。約180Lです。ドラム缶一杯分です。
17 一石=十斗=百升=千合です。
18 朝昼晩一合ずつご飯をいただくと、一年365日では約千合=一石になります。
19
20 お米を育てる田んぼの面積は次のような単位があります。
21 歩(ぶ)
22 歩は、一間（いっけん）四方の面積を表す単位です。一間は六尺で約182cmです。
23 歩は、一間四方なので畳二枚分の面積で、一坪と同じです。約3.3m2です。
24 畝(せ)
25 一畝は30歩、約99m2です。
26 おおよそ、住宅一軒分の面積です。
27 反(たん)
28 一反は10畝、300歩のことです。約990m2=約10a（アール）です。
29 農業の分野では反収として、よく用いられます。
30 反収（一反からの収穫量）は、北海道では約10俵、新潟では約9俵です。
31 一俵=四斗=60kgです。
32 町(ちょう)・一町分(ちょうぶ)
33 一町は10反で、約9900m2=約1ha（ヘクタール）です。
34
35 お米が取れたら、蔵にしまいます。大きな蔵、小さな蔵、いろいろありますが、
36 「一般的な土蔵は、桁行（けたゆき）四間（約7.2m）、梁間（はりま）二間半（約4.5m）」
37 、高さ18尺（5.4m）、二階建てで、建坪が10坪（33m2）、総延坪が20坪（66m2）です。
```

```

38 お米に関する頃知話もいろいろあります。
39 http://www.kumamotokokufu-h.ed.jp/kokufu/math/math\_7.html
40 https://minwanoheya.jp/area/yamagata\_022/
41 https://www.enshuryu.com/茶道具/落語-「-太閤と曾呂利」/
42
43 「盤上百万石 八十一枚の夢」
44 将棋が好きなお殿様、褒美をやることとします。
45 「なんでも望みの品を申せ」
46 「それではお言葉に甘えまして、将棋盤の最初の枚目には一粒のお米を、次の枚目には倍の>
   >二粒のお米を、更にその次の枚目には更に倍の四粒のお米をというように、最後の枚目まで>
   >米粒を賜りたく存じます」
47
48 さて、何粒のお米を賜ることができるでしょうか。
49 それは、何俵のお米で、何石になるでしょうか。
50 いくつ蔵を立てれば賜ったお米を納めることができるでしょうか。
51 非常に多い様を称して、「山ほど」のと形容しますが、富士山何座分でしょうか。
52 それだけのお米を作るためには、何町歩の田んぼが必要でしょうか。
53 =====*/
54
55 #include <stdio.h> // 標準入出力関数
56 #include <math.h> // 数学関数
57
58 int main(int argc, char const *argv[]) {
59
60     // 賜るお米の粒の数
61     //  $1 + 2 + 4 + 8 + \dots + 2^{80} = 2^{81} - 1$  粒 です。
62     //  $2 = 10^{0.3}$  ですので、
63     //  $2^{81} = 10^{(0.3 \times 81)} = 10^{24.3}$  なので、25桁の数になります。
64     // 25桁出力できるよう、書式指定子を%25ld とします。
65     printf("\n === 二の幂乗 === \n");
66     int n;
67     long g = 1; // 粒(grains)
68     for (n = 1; n <= 81; n++) {
69         g *= 2;
70         printf("%2d %25ld\n", n, g);
71     }
72
73     //  $2^1 = 2$  から始まり、
74     //  $2^{62} = 4611686018427387904$  までは正しく出力されています。
75     //  $2^{63} = -9223372036854775808$  と負号が付いています。
76     //  $2^{64}$  からは 0 と出力されます。
77     // long型は 64bit,  $-2^{63} \sim +2^{63}-1$ までの範囲の整数を表すことができます。
78     // この範囲を超えたため、出力結果が不正となりました。
79
80     // そこで、整数型による演算を手放し、浮動小数点数による演算を行うこととします。
81     // double型は  $2.225074 \cdot 10^{-308} < \text{double}$  の絶対値  $< 1.797693 \cdot 10^{308}$  までの
82     // 範囲の数を表現できます。
83     for (n = 1; n <= 81; n++) {
84         printf("%2d %28.2f\n", n, pow((double)2, (double)n));
85     }
86     // 二の幂乗を求めているため、とても正確に計算できています。
87     //  $2^{81}$  から 1を引いて 賜るお米の数を求めてみましょう。
88     for (n = 1; n <= 81; n++) {

```

```

89     printf("%2d %28.2f\n", n, pow((double)2, (double)n) - 1);
90 }
91 // 2のn乗は偶数です。そこから1を引いたので奇数となるはずですが、
92 // 途中からは偶数の値がoutputされています。
93 // double型の有効桁数は 二進数で53桁、十進数で約15桁です。
94
95 // 賜るお米の粒の数（近似値ですが、十分な精度です）
96 double grains = pow((double)2, (double)81) - 1;
97 // 升
98 double shou = grains / 64827;
99 // 斗
100 double to = shou / 10;
101 // 倔
102 double hyou = to / 4;
103 // 石
104 double koku = to / 10;
105 printf ("賜るお米の粒の数: %25.0f\n", grains);
106 printf ("          升: %25.0f\n", shou);
107 printf ("          斗: %25.0f\n", to);
108 printf ("          倔: %25.0f\n", hyou);
109 printf ("          石: %25.0f\n", koku);
110 // 37京2969兆8488億6378万4832石 と求めることが出来ました。
111
112 // 蔵の体積を求めます。蔵に米俵を隙間なく詰めると何俵入るでしょうか。
113 double kura = 7.2 * 4.5 * 5.4; // m³ (立方メートル)
114 double tawara = 180.0 * // 一合の体積は180ml
115           10 * // 一升
116           10 * // 一斗
117           4 / // 一俵
118           1000 / // L (リットル) 単位に変換
119           1000; // m³ (立方メートル) 単位に変換
120 double irisuu = kura / tawara;
121 double kurasuu = hyou / irisuu;
122 printf ("蔵の体積          : %25.3f\n", kura);
123 printf ("俵の体積          : %25.3f\n", tawara);
124 printf ("蔵に入る俵の数    : %25.3f\n", irisuu); // 一蔵2430俵入り
125 printf ("必要な蔵の数      : %25.3f\n", kurasuu);
126
127 // 一つの蔵には2430俵もの米俵を納めることが出来、
128 // 383兆7138億3627万9614 戸前もの蔵が必要になることが分かりました。
129
130 // 富士山の裾野はなだらかですので、富士山が始まる明瞭な境はありませんが、
131 // 麓にある富士山本宮浅間大社から富士山頂までの直線距離は、18.5kmです。
132 // そこで、半径18.5km 高さ3776m の円錐と看做して、富士山の体積を計算します。
133 double fujisan = 18500 * 18500 * 3.14 * 3776 / 3;
134 double irisuu_fujisan = fujisan / tawara;
135 double fujisansuu = hyou / irisuu_fujisan;
136 printf ("富士山の体積        : %25.3f\n", fujisan);
137 printf ("俵の体積          : %25.3f\n", tawara);
138 printf ("富士山に入る俵の数 : %25.3f\n", irisuu_fujisan);
139 printf ("必要な富士山の数   : %25.3f\n", fujisansuu);
140 // 富士山 49632座 分のお米の量になることが分かりました。
141

```

```

142 // 49632座の富士山がもし一つの山だったらどれくらいの大きさになるでしょうか。
143 // 長さの3乗が体積になりますので、
144 // 49632の立方根を取れば、巨大富士山の大きさを算出できます。
145 double bairitsu      = pow(fujisansuu, 1.0 / 3.0);    // 立方根
146 printf ("巨大富士山の裾野      : %25.3f\n", 18.5 * 2 * bairitsu);
147 printf ("巨大富士山の高さ      : %25.3f\n", 3.776 * bairitsu);
148 // 裾野の広がりは、1360kmと、青森から鹿児島までの直線距離に匹敵します。
149 // □ 頭を雲の上に出し・・・と歌われますが、
150 // 巨大富士山の高さは、139kmと宇宙まで届いています。
151
152 // 地球の赤道から北極までの長さは一万糠(キロメートル)です。
153 // ですので、地球の円周は40000km、直径12740km、半径6370km になります。
154 double menseki_chikyu = 4 * 3.14 * pow((double)6370, (double)2);
155 double menseki_fujisan = 3.14 * pow((double)18.5, (double)2);
156 printf ("地球の表面積      : %25.3f\n", menseki_chikyu);
157 printf ("富士山の底面積      : %25.3f\n", menseki_fujisan);
158 printf ("地表に占める割合(%) : %25.3f\n",
159           menseki_fujisan * 49632 / menseki_chikyu * 100);
160 // 49632座の富士山を地表に敷き詰めると、地球の表面積の一割強を占めます。
161 // 陸地は28.9%ですから、陸地の1/3以上が富士山で埋まることとなります。
162
163 // 反収 10 倍として、必要な田んぼの広さを計算します。
164 // 1 倍は 4 斗ですから、10 倍は 40 斗 = 4 石になります。
165 // 一反から 4 石のお米が取れることが分かりました。
166 // 賜るお米は 37京2969兆8488億6378万4832石 ですから、
167 // これを生み出すために必要な田んぼの広さを求めましょう。
168 double tan          = koku / 4;        // 田んぼの広さ 単位：反
169 double choubu       = tan / 10;       // 田んぼの広さ 単位：町歩 = 1ha
170 double km2          = choubu / 100;    // 田んぼの広さ 単位：km2
171 double nippon       = km2 / 378000;   // 田んぼの広さ 単位：日本
172 double chikyu       = km2 / 509645864; // 田んぼの広さ 単位：地球
173 printf ("反          : %21.0f\n", tan);
174 printf ("町歩         : %21.0f\n", choubu);
175 printf ("平方キロメートル : %21.0f\n", km2);
176 printf ("日本          : %21.0f\n", nippon);
177 printf ("地球          : %21.0f\n", chikyu);
178 printf ("太陽          : %21.0f\n", chikyu/109/109);
179
180 // 37京2969兆8488億6378万4832石は、
181 // 日本中を田んぼにして、2億4667万3180年前から今日まで、
182 // ずっとお米を作り続けていたら、採れることが分かりました。
183 // 二億五千万年前は三葉虫が大量絶滅した時代です。
184 // また、地球の表面全部を田んぼにできたとすると、
185 // 18万2955年前、ちょうど人類がアフリカを出たあたりのころです。
186 // 太陽の直径は約140万kmで、地球の直径の109倍ですが、
187 // もし太陽表面を全部田んぼにしてお米を育てたとすると、
188 // 15年、生まれたばかりの赤ちゃんが中学校を卒業するまでの期間です。
189 //
190 // 幕関数の凄さが分かりますね。
191 }

```

3.3 作成例 ★★★☆☆

♣ 定期預金

▼ deposit.c

```

1 #include <stdio.h> // 標準入出力関数
2
3 // お金が2倍になる期間を知る目安として、
4 // 「72の法則」が知られています。
5 // 金利3% なら  $72 \div 3 = 24$  年で 2倍になるという法則です。
6
7 int main(int argc, char const *argv[]) {
8     int n = 1; // 倍率
9     char buffer[80]; // sprintfのための文字列バッファ
10
11    for (n = 1; n <= 4; n++) {
12        double rate = 0.03; // 金利
13        double deposit = 1.0; // 定期預金
14        int year = 0; // 預けた年数
15
16        // 繰り返す回数が分からぬ場合は、while文を使います。
17        // 定期預金が2倍未満の間、繰り返します。
18        while (deposit < 2.0) {
19            // 自己代入演算子を使って
20            // deposit *= (1 + rate * n);
21            // と書くことも出来ます。
22            deposit = deposit * (1 + rate * n);
23            year++;
24        }
25
26        // printf の中で % を出力したい時には、%% と書きます。
27        printf("年利 %4.1f% の定期預金は %2d 年後に %5.3f 倍になりました。\n",
28               // 長くなった場合、カンマの後で改行することも出来ます。
29               (rate * 100 * n), year, deposit);
30
31        // sprintf 文は 書式指定した文字列を返す関数です
32        sprintf(buffer, "年利 %4.1f% の定期預金は %2d 年後に %5.3f 倍になりました.\n"
33               ,
34               (rate * 100 * n), year, deposit);
35        // 書式に従って、セットされた文字列を出力します。
36        printf("%s", buffer);
37    }
38
39    return 0;
}

```

♣ 千支を求める

▼eto2.c

```

1 #include <stdio.h> // 標準入出力関数
2
3 int main(int argc, char const *argv[]) {
4     int year = 2000; // グレゴリオ暦年
5
6     // 配列の各要素に、干支を格納しています。
7     // 12で割った余りが配列の添字になるよう、"申"から始めています。
8     // "申"と一文字に見えますが、
9     // 文字コードが utf-8 の場合、3文字(バイト) E7 94 B3 です。
10    // char saru[4] = { 'E7', '94', 'B3', '\n' };
11    // そして、干支は12種類ありますから、
12    // 配列の配列 (=二次元配列) にすれば、干支の配列になります。
13    char eto[][4] = {{"申", "酉", "戌", "亥", "子", "丑",
14                      "寅", "卯", "辰", "巳", "午", "未"}};
15    // 配列の一次元目の[4]は必要です。
16    // 4を書かずに、[]とだけ書くと、コンパイルエラーとなります。
17    // ポインタを使って次のように書くこともできます。
18    // char *eto[] = { "申", "酉", "戌", "亥", "子", "丑",
19                      "寅", "卯", "辰", "巳", "午", "未" };
20
21    int index;
22    index = year % 12;
23    printf("あなたの干支は、%s年 です。\\n", eto[index]);
24
25    // 一行に纏めて書くことも出来ます。
26    printf("あなたの干支は、%s年 です。\\n", eto[year % 12]);
27
28    return 0;
29 }
```

♣ 福引き

▼lottery.c

```

1 #include <stdio.h> // 標準入出力関数
2 #include <string.h> // 文字列操作関数
3 #include <stdlib.h> // rand関数
4 #include <time.h> // time関数
5
6 // 亂数はよく使うので、関数化してみました。
7 // 同じプログラムを何度も書かずに済むので、便利です。
8 // 自作したrandom_number()関数が定義されています。
9 // 自作のヘッダーファイルを読み込む際は、
10 // "(ダブルクォーテーション)"で囲みます。
11 #include "random_number.h"
12
13 int main(int argc, char const *argv[]) {
14     // 福引きの等級名
15     // 配列の添字は0から始まるので、先頭に""を入れています。
16     // prize_rank[1] が "一等賞"だと分かりやすいです。
17     char *prize_rank[] = {"", "一等賞", "二等賞", "三等賞", "残念賞"};
18     // 福引きの賞品
```

```

19 char *prize_item[] = {"", "世界一周の旅", "温泉一泊二日",
20                               "お好み焼き食べ放題", "ティッシュペーパー"};
21 // メッセージ
22 char message[64];
23
24 // 亂数はよく使うので、流用しています。
25 int lottery = random_number(10) + 1; // 福引き 1-10の乱数が得られます。
26
27 int rank; // 何等賞か？
28 if (lottery == 1) {
29     rank = 1; // 一等賞
30 } else if (2 <= lottery && lottery <= 3) {
31     rank = 2; // 二等賞
32 } else if (4 <= lottery && lottery <= 6) {
33     rank = 3; // 三等賞
34 } else {
35     rank = 4; // 残念賞
36 }
37
38 // 小さい順に切り取っているので、
39 // else if の下限は省略出来ます。
40 if (lottery == 1) {
41     rank = 1; // 一等賞
42 } else if (lottery <= 3) {
43     rank = 2; // 二等賞
44 } else if (lottery <= 6) {
45     rank = 3; // 三等賞
46 } else {
47     rank = 4; // 残念賞
48 }
49
50 // ちなみに条件式を書けないので、switch case 文には出来ません。
51
52 // それぞれの等級に応じたメッセージを創っています。
53 // 練習のために、文字列操作の関数を使っています。
54 strcpy(message, "おめでとう！"); // コピー関数
55 strcpy(message, prize_rank[rank]); // コピー関数
56 strcat(message, ":"); // 連結関数
57 strcat(message, prize_item[rank]); // 連結関数
58 strcat(message, " が当たったよ。"); // 連結関数
59 printf("%s\n", message);
60
61 return 0;
62 }
```

♣ 開年

▼ leap_year.c

```

1 #include <stdio.h> // 標準入出力関数
2 #include <math.h> // 数学関数
3 #include <string.h> // 文字列操作関数
4 #include <stdlib.h> // rand関数
```

```

5 // 分かりやすさのために、
6 // 真であることを示すTRUE,
7 // 偽であることを示すFALSE という定数を定義します。
8 // 定数は全て大文字で書く慣習です。
9
10 #define TRUE 1
11 #define FALSE 0
12
13 // 関数のプロトタイプ宣言
14 // グレゴリオ暦年を渡して、閏年なら、TRUEを返す関数
15 // (どちらも閏年を返す関数ですが、内部の実装が異なります)
16 int leap_year1(int year);
17 int leap_year2(int year);
18
19 int main(int argc, char const *argv[]) {
20     // 使い方の説明表示
21     // コマンドラインから、引数を渡すことも出来ます。
22     if (argc == 1) {
23         printf("【使い方】\n");
24         printf(
25             "閏年(leap year)か、平年(common year)か、算出するプログラムです。\n");
26         printf("%s 2030\n", argv[0]); // argv[0] はプログラム自身の名前です。
27         printf("の様にグレゴリオ暦年で入力して下さい。.\n");
28
29         exit(1); // プログラムを終了します。
30     }
31
32     int year;      // グレゴリオ暦年
33     char *endptr; // 数値に変換出来なかった文字
34     // atoi関数は、文字列を数値に変換出来なかった場合にでも、0を返します。
35     // 本当に、「0」という文字列が、0という数値に変換されたのか、
36     // 区別が出来ないので、strtol関数を使います。
37     year = strtol(argv[1], &endptr, 10); // 10進数として変換する
38
39     // 数値変換不可の文字列の長さを調べる。
40     if (strlen(endptr) != 0) {
41         // エラーメッセージ出力
42         printf("%s は、グレゴリオ暦年として認識出来ませんでした。.\n", argv[1]);
43         exit(1); // エラーコード 1 として、異常であることを伝えて終了します。
44     }
45
46     // 閏年かどうかはよく使うので、自作の関数を創って、判定することにします。
47     // leap_year1 : 長いif文の関数
48     if (leap_year1(year) == TRUE) {
49         printf("閏年です。.\n");
50     } else {
51         printf("平年です。.\n");
52     }
53
54     // leap_year2 : 整理したif文の関数
55     if (leap_year2(year) == TRUE) {
56         printf("閏年です。.\n");
57     } else {

```

```
58     printf("平年です。\\n");
59 }
60
61     return 0;
62 } // main()関数の最後です。
63
64 // グレゴリオ暦年を渡して、閏年なら、TRUEを返す関数
65 int leap_year1(int year) {
66     // 4で割り切れる年は閏年です。
67     // 但し100で割り切れる年は閏年ではありません。
68     // しかしながら、400で割り切れる年は、閏年です。
69
70     // 素直にif文で書くと次のようにになります。
71     // (if文の中にif文を書くことも出来ます。)
72     if (year % 4 == 0) {
73         // 4で割り切れる年は閏年ですが、例外があるので、例外を書きます。
74         if (year % 100 == 0) {
75             // 但し100で割り切れる年は閏年ではありません。とあります。
76             // さらにこれの例外があるので、例外を書きます。
77             if (year % 400 == 0) {
78                 // しかしながら、400で割り切れる年は、閏年です。とあるので、
79                 // TRUEを返します。
80                 return TRUE;
81             } else {
82                 // 400で割り切れなかった年です。
83                 return FALSE;
84             }
85         } else {
86             // 100で割り切れなかった年です。
87             return TRUE;
88         }
89     } else {
90         // 4で割り切れなかった年です。
91         return FALSE;
92     }
93 }
94
95 // グレゴリオ暦年を渡して、閏年なら、TRUEを返す関数
96 int leap_year2(int year) {
97     // 4で割り切れる年は閏年です。
98     // 但し100で割り切れる年は閏年ではありません。
99     // しかしながら、400で割り切れる年は、閏年です。
100
101    // 素直にif文を書くとすごく長くなってしまいました。
102    // if else が複雑になっていて、合っているのか間違っているのか、
103    // 確認するのも大変です。
104
105    // 4で割り切れる.....(a)
106    // 100で割り切れる.....(b)
107    // 400で割り切れる.....(c)
108    // と条件が複合しているので、
109    // 閏年かどうか.....(x)
110    // 表にして整理すると、分かりやすいです。
```

```

111 // ( 真偽値表 、カルノー図と言います )
112
113 // 割り切れることを TRUE(T)
114 // 割り切れないことを FALSE(F) として、表にしてみましょう。
115 // 8 とおりの組み合わせが出来ます。
116 // - が入っているところは、あり得ない組み合わせのところです。
117 // 4 で割り切れなければ、当然、100 でも 400 でも割り切れませんものね。
118
119 // a b c x
120 // F - - F ... (1)
121 // F - - F ... (2)
122 // F - - F ... (3)
123 // F - - F ... (4)
124 // T F - T ... (5)
125 // T F - T ... (6)
126 // T T F F ... (7)
127 // T T T T ... (8)
128
129 // 表を見ると、(5) と (6) は同じですので、
130 // (5) または (8) の場合で、閏年になることが分かります。
131 // つまり、
132 // 4で割り切れて、100で割り切れない年
133 // または、
134 // 4で割り切れて、100で割り切れて、400で割り切れる年(=400で割り切れる年)
135 // の場合に閏年になることが分かります。
136
137 // よって次のif文でよいことが分かります。
138 // (論理積演算子&&は、論理和演算子||より、優先順位が高いので
139 // ()は不要ですが、分かりやすさの為に、()を付けています。)
140 if ((year % 4 == 0 && year % 100 != 0) || year % 400 == 0) {
141     return TRUE;
142 } else {
143     return FALSE;
144 }
145 // 1600, 1700, 2000, 2004, 2099, 2100年を入れて、確認してみましょう。
146 // (どのテストケースで確認すべきか、考えるのも大切です)
147 }
```

♣ お天気予報

▼ umbrella2.c

```

1 #include <stdio.h> // 標準入出力関数
2 #include <stdlib.h> // atoi関数
3
4 int main(int argc, char const *argv[]) {
5     // コマンドラインから、降水確率を渡すことも出来ます。
6
7     // 使い方の説明表示
8     // (argv[0] はプログラム自身の名前です。)
9     if (argc == 1) {
10         printf("【 使い方 】\n");
11         printf("傘を持っていくべきか、助言するプログラムです。\\n");
```

```

12 printf("もし降水確率が 30% なら\n");
13 printf("%s 30\n", argv[0]);
14 printf("と入力して下さい。.\n");
15
16 exit(1); // プログラムを終了します。
17 }
18
19 // atoi関数は、文字列を数値に変換する関数です。
20 // argv[1]が 文字列 "30" なら、
21 // precipitation_probability には、整数 30 が入ります。
22 int precipitation_probability = atoi(argv[1]);
23
24 if (precipitation_probability <= 30) {
25     printf("傘はいらないです\n");
26 } else if (precipitation_probability <= 70) {
27     printf("持って行った方がいいかも\n");
28 } else {
29     printf("絶対持って行きましょう\n");
30 }
31
32 return 0;
33 }
```

♣ 白銀比

▼silver_ratio.c

```

1 =====
2 【白銀比】
3 長方形を二つ折りにした時にできる長方形が、元の長方形と相似である時の
4 長辺と短辺の比を「白銀比」と言います。
5      x
6      -----
7 1 |     |     | 1
8  |     |     |
9  -----
10        x/2
11
12 大きな長方形は、長辺 x : 短辺 1
13 二つ折りにした小さな長方形は、長辺 1 : 短辺 x/2
14 これが相似なので、
15 x : 1 = 1 : x/2
16 が成り立ちます。
17 外項の積は内項の積に等しいので
18 x(x/2) = 1
19 x^2 = 2
20 x = √2 と求められます。
21
22 二つ折りにして元の紙と相似という性質は、印刷製本の際にとても便利です。
23 紙の大きさの規格として、A3, A4 用紙、B4, B5 用紙をよく見かけますが、
24 A3用紙を二つ折りにするとA4用紙、B4用紙を二つ折りにするとB5用紙になります。
25 また、A4用紙の面積を1.5倍にしたもののがB4用紙です。
26
```

```

27 A0 用紙の面積を1m2平方メートル)として、
28 それを二つ折りにしたものがA1, それをさらに二つ折りにしたものがA2と、
29 A0, A1, A2, ... A10 まで規格されています。
30 それぞれの紙の大きさを求めてみましょう。
31 (ミリメートル単位で求め、端数を切り捨てます。)

32
33 【参考】
34 https://www.tcpc.co.jp/columns/index049
35 =====*/
36
37 #include <stdio.h> // 標準入出力関数
38 #include <math.h> // 数学関数
39
40 // 短辺と長辺を与えると、二つ折りにした紙の長さを返す関数
41 void half_swap(int *short_side, int *long_side);
42
43 int main(int argc, char const *argv[]) {
44     int n = 0; // A0版, A1版, A2版 ... A10版 まで
45     int short_side; // 短辺
46     int long_side; // 長辺
47
48     // A0版の面積は1m2(平方メートル)です。
49     // 面積が1m2ですから、概ね一辺1mくらいと察しができます。
50     // 長辺の長さは短辺の長さの $\sqrt{2}$ 倍( $= 2^{0.5}$  2 の0.5乗>)
51     // という関係がありますから、
52     // -0.25 0.25 0
53     // 2 × 2 = 2
54     // 短辺 長辺 面積
55     // であることが分かります。(指數の法則を思い出してください。)
56
57     // 短辺の長さ
58     short_side= (int)round(pow(2, -0.25) * 1000); // mm 単位にして、
59                             // round関数で端数を四捨五入。
60                             // 整数型(int)へ型変換します。
61
62     // 長辺の長さ
63     long_side = (int)round(pow(2, 0.25) * 1000);
64
65     for (n = 0; n <= 10; n++) {
66         printf("A%d版 短辺 %d mm 長辺 %d mm\n", n, short_side, long_side);
67         half(&short_side, &long_side);
68     }
69
70     return 0;
71 }
72
73 void half_swap(int *short_side, int *long_side) {
74     double work; // 作業用変数
75
76     work = *short_side;
77     *short_side = *long_side / 2; // 整数型演算の為、端数は捨てられます。
78     *long_side = work;
79 }
```

♣ 単語の長さ

▼ word_length.c

```

1  ****
2  * 入力された英単語の長さを表示する
3  * ( https://nzlife.net/archives/9581 に長い英単語の豆知識があります )
4  ****
5 #include <stdio.h> // 標準入出力関数
6 #include <stdlib.h> // rand関数
7 #include <string.h> // 文字列操作関数
8
9 // #define で、TRUE という定数を1であると定義しています
10 // プログラム中によく使う定数は、このように定義しておくと、
11 // 意味が分かりやすくて、よいです。
12 #define TRUE 1
13
14 int main(int argc, char const *argv[]) {
15     char buffer[64]; // 英単語の読み込み用
16
17     while (TRUE) {
18         // 入力を促すメッセージの表示
19         printf("英単語を入力して下さい。(終了:bye)\n");
20
21         // ちなみにプログラム学習用なので、
22         // scanf("%s", buffer);
23         // の一行が簡単だったりします。
24
25         // キーボードから一行読み込む
26         // fgets関数で、bufferへ、sizeof(buffer)-1文字分(7文字分),
27         // stdin(=キーボード)から読み込む
28         if (fgets(buffer, sizeof(buffer), stdin) == NULL) {
29             // エラーメッセージ出力
30             printf("キーボードから読み込めませんでした。\n");
31             exit(1);
32         }
33
34         // 改行文字が含まれているかどうか？
35         if (strchr(buffer, '\n') != NULL) {
36             // cakeエンター のように5文字タイプされたときは、
37             // 改行文字(エンター)を文字列の終端記号に置換する
38             buffer[strlen(buffer) - 1] = '\0';
39         } else {
40             // buffer内に、改行文字が含まれていない場合 (=8文字以上続けてタイプされた場合)
41             // 最初の7文字は読み込まれているので、残りの入力ストリーム(キーバッファ)を
42             // クリアする
43             while (getchar() != '\n')
44                 ;
45         }
46
47         // 無限ループとなっているので、
48         // プログラム終了のための文字列 "bye" と比較します。
        // strcmp は 比較した文字が小さいとき(辞書順に並べたときに前にくる場合) -1

```

```

49 // を strcmp は 比較した文字が大きいとき(辞書順に並べたときに後に行く場合)
50 // 1 を返します。そして、C言語では、0以外は真と判断しますので、if
51 // ((strcmp(buffer, "bye"))){ と書いても同じですが、!= 0
52 // と明示されていると、分かりやすいかと思います。
53 if ((strcmp(buffer, "bye")) != 0) {
54     printf("入力された英単語は、%s ですね。\n", buffer);
55     printf("長さは、%lu 文字の単語ですね。\n", strlen(buffer));
56     printf("意味は・・・ う～ん分かりません。\n");
57     printf("\n"); // 前の行に\nを二つ入れてもOKですが、
58                 // 画面表示と見た目をあわせて置いた方が分かりやすいです。
59 } else {
60     printf("また、使ってね。bye-bye\n");
61     break;
62 }
63 }
64
65 return 0;
66 }
```

♣ 縦棒グラフ

▼ vertical_bar_graph.c

```

1 #include <stdio.h> // 標準入出力関数
2 #include <string.h> // 文字列操作関数
3
4 // グラフのX軸、Y軸の上限を定義
5 #define X_MAX 5
6 #define Y_MAX 7
7
8 int main(int argc, char const *argv[]) {
9
10    // 縦棒グラフを表示する
11    // y
12    // ^
13    // | □□■□□
14    // | □□■□□
15    // | □□■□□
16    // | □■■□□□
17    // | □■■■□□
18    // | □■■■■□
19    // | ■■■■■■
20    // -----> x
21
22    // 表示させたい棒グラフの値
23    int values[] = {1, 4, 7, 3, 2};
24    // グラフ用の二次元配列
25    char graph[X_MAX][Y_MAX][6]; // "■", "□" は、UTF-8 では E2 96 A0, E2 96 A1 と
26                                // 6文字で表現される為 [6] としている。
27    // char graph[X_MAX][Y_MAX]; // '*', 'o' の一文字で表示するなら、これで良い。
28
29    // graph[0]
30    // □ graph[0][6]
```

```

31 // □ graph[0][5]
32 // □ graph[0][4]
33 // □ graph[0][3]
34 // □ graph[0][2]
35 // □ graph[0][1]
36 // ■ graph[0][0]
37 // のようにしたい。
38
39 // value の値に応じて、棒グラフの長さをセットする
40 int x, y;
41 for (x = 0; x < X_MAX; x++) {
42     int value = values[x];
43     for (y = 0; y < Y_MAX; y++) {
44         if (y < value) {
45             // graph[x][y] = "■"; // こう書きたいが、書けないので strcpyを用いる。
46             // graph[x][y] = '*'; // '*'のように char型一文字なら = で代入できる。
47             strcpy(graph[x][y], "■");
48         } else {
49             strcpy(graph[x][y], "□");
50         }
51     }
52 }
53
54 // グラフ表示
55 for (y = Y_MAX-1; y >= 0; y--) {
56     for (x = 0; x < X_MAX; x++) {
57         printf("%s", graph[x][y]);
58         // '*', 'o' の一文字で表示するなら、%s に代えて %c を用いると良い。
59         // printf("%c", graph[x][y]);
60     }
61     printf("\n");
62 }
63
64 return 0;
65 }
```

♣ 計算ゲーム

▼ calculation_game2.c

```

1 ****
2 * 足し算ゲームとほぼ同様です。
3 * 桁数と、演算の種類をリクエストするようになっています。
4 * 簡単なものを創って、より高機能のものへと、拡張していくと良いです。
5 ****
6 #include <stdio.h> // 標準入出力関係です。
7 #include <math.h> // 数学関係の関数が定義されています。
8 #include <string.h> // 文字列操作に関する関数が定義されています。
9 #include <stdlib.h> // rand関数が定義されています。
10 #include <time.h> // time関数が定義されています。
11 // 自作したrandom_number()関数が定義されています。
12 // 自作のヘッダーファイルを読み込む際は、
13 // "(ダブルクォーテーション)"で囲みます。
```

```

14 #include "random_number.h"
15
16 int main(int argc, char const *argv[]) {
17     int operand1;          // 第一被演算子
18     int operand2;          // 第二被演算子
19     char operator;         // 演算子 ( '+', '-', '*', '/' )
20     int digit;            // 桁数
21     int result;           // 演算結果
22     int answer;           // 回答
23     int correct;          // 正答数
24     int n;                // 繰り返し回数
25
26     // 桁数をリクエスト
27     do {
28         // 文字を入力されたときは、無視します。
29         printf("何桁の数字で挑戦しますか？\n");
30         printf("一桁: 1 二桁: 2 三桁: 3 四桁: 4\n");
31         scanf("%d", &digit);
32         while (getchar() != '\n')
33             ; // キーバッファ読み飛ばす
34     } while (digit <= 0 || digit >= 5);
35
36     // 演算子をリクエスト
37     do {
38         printf("どの演算に挑戦しますか？\n");
39         // 足し算、引き算、掛け算、割り算のことです。
40         // 加算、減算、乗算、除算という呼び方も知っておいて欲しいです。
41         printf("加算: + 減算: - 乗算: * 除算: /\n");
42         scanf("%c", &operator);
43         while (getchar() != '\n')
44             ; // キーバッファ読み飛ばす
45         // char型 一文字なので、簡単に比較出来ます。
46         // 全角で"+"や"-"なら文字列操作関数を使って下さい。
47     } while (operator != '+' &&
48             operator != '-' &&
49             operator != '*' &&
50             operator != '/');
51
52     correct = 0; // 10問中何問正解か数えるために、初期化します。
53     // 1回目、2回目と表示させたいので、
54     // for (n = 1; n <= 10; n++) と書いているところに着目して下さい。
55     for (n = 1; n <= 10; n++) {
56         // 出題処理
57         // 幕乗を求める関数を使って、必要な桁数に調整します。
58         operand1 = random_number(pow(10, digit));
59         // do while文で、異なる数になるまで、繰り返します。
60         do {
61             operand2 = random_number(pow(10, digit));
62             // 0の割り算は定義されていないので、条件式を変更します。
63         } while (operand2 == operand1 || (operator == '/' && operand2 == 0));
64
65         switch (operator) {
66             // switch case文には、文字型も使えます。

```

```
67     case '+':
68         result = operand1 + operand2;
69         break;
70     case '-':
71         result = operand1 - operand2;
72         break;
73     case '*':
74         result = operand1 * operand2;
75         break;
76     case '/':
77         result = operand1 / operand2;
78         break;
79     }
80
81 // ゲーム名を表示するための場合分けです。
82 // こう書いた方が素直です。
83 // switch(operator){
84 // case '+':
85 //     printf("足し算ゲーム %d 回目", i);
86 //     break;
87 // case '-':
88 //     printf("引き算ゲーム %d 回目", i);
89 //     break;
90 // case '*':
91 //     printf("掛け算ゲーム %d 回目", i);
92 //     break;
93 // case '/':
94 //     printf("割り算ゲーム %d 回目", i);
95 //     break;
96 // }
97
98 // char型が0-127までの数字として扱われることに着目した書き方
99 // 連想配列(ハッシュ)のご紹介
100 //
101 // ASCIIコード表を見ると、
102 // '+' 43
103 // '-' 45
104 // '*' 42
105 // '/' 47
106 // なっていますので、
107 // 47個+1個の大きさの文字列の配列を用意します。
108 char game_name[47 + 1][20];
109 strcpy(game_name['+'], "足し算");
110 // 書いて、配列の初期値を設定します。
111 // strcpy(game_name[43], "足し算");
112 // と書いたのと同じです。
113 strcpy(game_name['-'], "引き算");
114 strcpy(game_name['*'], "掛け算");
115 strcpy(game_name['/'], "割り算");
116 // 実際に printf("%s", game_name['+']);と書くと「足し算」と表示されます。
117 // このことをわきまえると、
118 printf("%sゲーム %d 回目\n", game_name[operator], n);
119 // と書けます。
```

```

120 // 配列は、添字として、数字をとりましたが、
121 // 文字を添字として使えるようになると、分かりやすく便利です。
122 // game_name["+"] = "足し算";
123 // の様なイメージです。
124 // 連想配列（ハッシュ）として、用意されている言語もあります。
125 // 他言語学習の際の参考になさって下さい。
126
127 printf("%d %c %d = ?\n", operand1, operator, operand2);
128
129 // 回答を受け取る
130 scanf("%d", &answer);
131 while (getchar() != '\n')
132     ; // キーバッファ読み飛ばす
133
134 // 正解発表と正答数のカウント
135 if (answer == result) {
136     printf("正解です。\n");
137     correct++;
138 } else {
139     printf("正解は、%d です。\n", result);
140 }
141
142 }
143
144 // 総合結果発表
145 printf("10問中 %d 問 正解\n", correct);
146
147 return 0;
148 }
```

♣ 配列の要素の並び替え

▼ selection_sort.c

```

1 #include <stdio.h> // 標準入出力関数
2 #include <limits.h> // 整数の取り得る範囲
3
4 int main(int argc, char const *argv[]) {
5     // 10個の数字が入った並び替え前の配列
6     int array[] = {3, 15, 22, 81, 41, 83, 72, 0, 50, 33};
7     int min; // 配列の最小値
8     int index; // 最小値の添字
9     int i;
10
11    // 最小値を求めると同時に、
12    // 最小値 min が何番目の要素であるか求める処理
13    min = INT_MAX;
14    index = 0;
15    for (i = 0; i < 10; i++) {
16        if (min > array[i]) {
17            min = array[i]; // (a) 最小値を求める
18            index = i; // (b) 添字を求める
19        }
}
```

```

20 }
21
22 // 最小値 min が 0 で、
23 // その添字 index は 7 と判明しましたので、
24 // 配列の一一番最後に 0 を入れることにします。
25 // そのまま入れると最後の値 33 が消えてしまうので、
26 // 後ろの要素を前の要素に詰めてから、
27 // 最後に0を入れることにします。
28 // {3, 15, 22, 81, 41, 83, 72, 0, 50, 33}
29 //           / \
30 //           ↓   ↓
31 // {3, 15, 22, 81, 41, 83, 72, 50, 33, 0}
32 //           ↑
33 //                         最小値を最後に入れる
34
35 // array[7] に array[8] を代入して、
36 // array[8] に array[9] を代入します。
37 for (i = index; i < 9; i++) {
38     array[i] = array[i + 1];
39 }
40 // array[9] に今求めた一番小さい数を入れます。
41 array[9] = min;
42
43 // 確認の為に出力してみましょう。
44 printf("一番小さい数が、最後に来ていることの確認\n");
45 for (i = 0; i < 10; i++) {
46     printf("array[%d]: %2d \n", i, array[i]);
47 }
48
49 // これで、一番小さい数を、最後に持って行くことが出来ました。
50 // 次は、0番目～8番目で並び替え、
51 // その次は、0番目～7番目で並び替え、
52 // ということを順番に繰り返していくと、全部の並び替えが完了します。
53
54 // 完成版のプログラムです。
55 // 0番目～何番目までを並び替えるのか、
56 // 管理するためのfor文で外側をくるんでいます。
57 // この並び替えのアルゴリズムを、選択ソート と言います。
58
59 int sorted; // 今、何番目の要素まで並び替えが終了しているのか。
60 for (sorted = 9; sorted >= 1; sorted--) {
61     min = INT_MAX;
62     for (i = 0; i <= sorted; i++) {
63         if (min > array[i]) {
64             min = array[i];
65             index = i;
66         }
67     }
68     for (i = index; i < sorted; i++) {
69         array[i] = array[i + 1];
70     }
71     array[sorted] = min;
72 }

```

```

73
74     printf("\n");
75     printf("【選択ソート】並び替え結果\n");
76     for (i = 0; i < 10; i++) {
77         printf("array[%d]: %2d \n", i, array[i]);
78     }
79
80     // 選択ソートの他に、有名な並び替え方法としては、
81     // シェルソート
82     // クイックソート
83     // マージソート
84     // などがあります。
85     // どうぞ学習なさって下さい。
86
87     return 0;
88 }
```

♣ 石取りゲーム

▼ stone_game.c

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4
5 // 定数定義
6 #define MAX      3    // 一度に石を取れる最大数
7 #define PLAYER   0
8 #define COMPUTER 1
9
10 const char *NAMES[] = {"プレーヤー", "コンピュータ"};
11
12
13 // プレーヤーの取った石の数
14 int capture_stone(char turn, int stone) {
15     int n; // 取る石の数
16     switch(turn){
17         // プレーヤーの番なら、取りたい個数を入力する
18     case PLAYER:
19         do {
20             printf("石を何個取りますか(1-3)\n");
21             scanf("%d", &n);
22             while (getchar() != '\n')
23                 ;
24         } while (!(1 <= n && n <= 3 && n <= stone)); // その場にある石の数を超えない
25         break;
26
27         // コンピュータの番なら
28     case COMPUTER:
29         // 4個以下の時には一つ残して取る
30         if (stone <= 4) {
31             n = stone - 1;
32         } else {
```

```

33     // 亂数で適当に取る
34     srand(time(NULL));
35     n = rand() % 3 + 1;
36 }
37 break;
38 }
39
40 return n;
41 }

// 石の表示
43 void disp_stone(int stone, int stone_max) {
44     int i;
45     for (i = 1; i <= stone; i++) { printf("●"); }
46     for (i = stone + 1; i <= stone_max; i++) { printf("○"); }
47     printf("\n");
48 }
49 }

int main(int argc, char const *argv[]) {
// 変数宣言
53     int stone_max; // 場にある石の最大数
54     int stone;      // 場にある石の数
55     int n;          // 競技者が取った石の数
56     char turn;      // どちらの番か？

57     // 亂数で石の総数を決めます
58     srand(time(NULL));
59     stone_max = rand() % 5 + 20;
60     stone      = stone_max;

62     // オープニング
63     printf("===== 石取りゲーム =====\n");
64     printf("交互に 1～3 個の石を取ります。\n");
65     printf("最後の一ヶを取ると負けです。");
66     printf("最初は %d 個の石があります。", stone);

68     // 先攻後攻
69     printf("先攻しますか？ Y/N\n");
70     scanf("%c", &turn);
71     while (getchar() != '\n')
72         ;
73     if (turn == 'Y' || turn == 'y') {
74         // turn は char型 = -128~127までの整数型 なので、PLAYERを代入できます
75         turn = PLAYER;
76         printf("プレーヤーの先攻です\n");
77     } else {
78         turn = COMPUTER;
79         printf("コンピュータの先攻です\n");
80     }
81     printf("\n===== ゲーム開始 ===== \n");

84     // 石を交互にとる
85     do {

```

```
86 printf("\n%sの番です\n", NAMES[turn]);
87 printf("%d 個の石が残っています\n", stone);
88 disp_stone(stone, stone_max);
89 n = capture_stone(turn, stone);
90 printf("%d 個の石を取りました。\n", n);
91 stone -= n;
92 turn = (turn+1) % 2; // 交代
93 } while (stone > 0);
94
95 // エンディング
96 printf("\n%sの勝ちです\n", NAMES[turn]);
97 }
```

3.4 作成例 ★★★★☆

♣ 成績発表

▼ score_and_name_sort.c

```

1  ****
2  * 10人の名前が格納された配列と、10人の点数が格納された配列があります。
3  * 成績の良い順に名前と点数を表示してみましょう。
4  *
5  * C言語には、クイックソート(quicksort)と呼ばれるアルゴリズムで
6  * 実装された関数が標準で用意されていますので、今回はこれを用います。
7  * (前回、作成したプログラムを流用してももちろんOKです)
8  *
9  * クイックソートに関しては、
10 * https://ja.wikipedia.org/wiki/クイックソート
11 * を参照して下さい。
12 *
13 * qsortの利用方法については、
14 * http://www.cc.kyoto-su.ac.jp/~yamada/ap/qsort.html より、引用
15 ****
16 #include <stdio.h> // 標準入出力関数
17 #include <stdlib.h> // quicksort関数
18 #include <string.h> // 文字列操作関数
19
20 // 比較関数(大小判断を返す関数)
21 int compare_int(const void *a, const void *b) {
22     // b > a なら 正の数を返す。(降順)
23     return *(int *)b - *(int *)a;
24 }
25
26 int main(int argc, char const *argv[]) {
27     // 変数宣言
28     char *name[] = {"亜希", "加世", "小夜", "多実", "奈美",
29                     "太郎", "次郎", "三郎", "四郎", "五郎"};
30     int score[] = {55, 88, 77, 66, 55,
31                   55, 64, 73, 82, 91};
32     int backup[10];
33     int i, j, s;
34
35     // 結果表示
36     printf("並び替え前\n");
37     for (i = 0; i < 10; i++) {
38         printf("%s %d\n", name[i], score[i]);
39     }
40
41     // 並び替えすると、元の配列が破壊されるため、バックアップを取る
42     // for文で一要素ずつコピーしても良いが、
43     // memcpy関数を紹介
44     memcpy(backup, score, sizeof(int) * 10); // int型10要素分をbackupへコピー
45
46     // 確認用
47     // printf("score backup\n");

```

```

48 // for (i = 0; i < 10; i++){
49 //   printf("%d %d \n", score[i], backup[i]);
50 // }
51
52 // 並び替えを行う
53 // 並び替えたい配列名、要素数、配列1要素分のサイズ、大小比較に用いる関数名
54 qsort(score, 10, sizeof(int), compare_int);
55
56 // 確認用
57 // printf("並び替え後:\n");
58 // for (i = 0; i < 10; i++){
59 //   printf("%d\n", score[i]);
60 // }
61
62 printf("並び替え後\n");
63 for (i = 0; i < 10; i++) {
64   printf("%d \n", score[i]);
65 }
66
67 // 並び替え結果に基づいて、名前を表示する
68 for (i = 0; i < 10; i++) {
69   s = score[i];
70   // 例) i = 0 のとき s には 91 点が入っている
71   // 名前も運動して並び替えられると良いが、
72   // そういう作りにはしていないので、
73   // backup配列を参照して、
74   // 91 点の人は何番目であったか、検索する
75   for (j = 0; j < 10; j++) {
76     if (s == backup[j]) {
77       // 同じ点数（亜希、奈美、太郎）の場合、
78       // 既に選択済みであることを示す為、
79       // あり得ない点数の -1 を設定する。
80       // （「番兵」と呼ばれるテクニックです。）
81       backup[j] = -1;
82       break; // 元の順番では、j 番目であったことが分かった
83     }
84   }
85   printf("%s %d \n", name[j], score[i]);
86 }
87
88 return 0;
89 }

```

♣ 世界人口

▼ world_population.c

```

1 #include <stdio.h> // 標準入出力関数
2 #include <string.h> // 文字列操作関数
3
4 int main(int argc, char const *argv[]) {
5
6   long population = 8000000000; // 世界人口 80億人

```

```

7 // 約21億以上の数は、long型を使います。
8 double growth_rate = 0.01;           // 人口増加率 1%
9
10 // 来年の人口
11 // int, double, long
12 // それぞれの型が自動的に変換されていることに着目して下さい。
13 long next_year_population = population * (1 + growth_rate);
14 // long型の書式指定子は、%ld です。
15 printf("来年の人口は %ld 人です。\\n", next_year_population);
16
17 // 何年後に100億人になるか、求めてみましょう。
18 // 繰返し回数が不明な場合は、while文を使います。
19 int year = 0;
20 while (population < 1e10) { // 1e10 は、10の10乗(=100億)です。
21     population *= (1 + growth_rate);
22     year++;
23     printf("%d 年後の人口は %ld 人です。\\n", year, population);
24 }
25
26 // 3桁ごとにカンマ区切りで出してみましょう。
27 // 数としての人口を文字列に変換します。
28 char str_population[32];           // 世界人口が入った文字列
29 char str_population_reverse[32]; // 逆順になっている文字列（作業用）
30 char str_comma_population_reverse
31     [32]; // カンマを入れて逆順になっている文字列（作業用）
32 char str_comma_population[32]; // カンマを入れた文字列
33 sprintf(str_population, "%ld", next_year_population);
34 int digit = strlen(str_population);
35 printf("%d 衡の数です。\\n", digit);
36
37 // 0123456789
38 // str_population : 8000000000
39 // str_comma_population : 8,000,000,000
40
41 // カンマは、最後の桁から3つごとに入れていきます。
42 // 最後から数えて3つずつカンマを入れていくのは、
43 // 扱いにくいので、先頭から3つずつ入れていけば良いように、
44 // 逆順に並び替えます。
45 // 8000000000 -> 0000000008
46 int i;
47 for (i = 0; i < digit; i++) {
48     str_population_reverse[digit - i - 1] = str_population[i];
49 }
50 // 文字列の終わりが分かるように最後に'\\0'を入れます。
51 str_population_reverse[digit] = '\\0';
52 printf("逆順に並び替えて、%s となりました。\\n", str_population_reverse);
53
54 // 3桁ごとにカンマを入れていきます。
55 int comma = 0; // 今までに入れたカンマの数
56 for (i = 0; i < digit; i++) {
57     str_comma_population_reverse[i + comma] = str_population_reverse[i];
58     // 0桁目、1桁目では何もしませんが、
59     // 2桁目のコピーが終わった後に、

```

```

60 // カンマを追加する処理を行います。
61 if (i % 3 == 2) {
62     comma++;
63     str_comma_population_reverse[i + comma] = ',';
64 }
65 }
66 // 文字列の終わりが分かるように最後に'＼0'を入れます。
67 str_comma_population_reverse[digit + comma] = '\0';
68 printf("カンマを追加して、%s となりました。\n", str_comma_population_reverse);
69
70 // もう一度、並び替えます。
71 for (i = 0; i < digit + comma; i++) {
72     str_comma_population[digit + comma - i - 1] =
73         str_comma_population_reverse[i];
74 }
75 // 文字列の終わりが分かるように最後に'＼0'を入れます。
76 str_comma_population[digit + comma] = '\0';
77 printf("もう一度並び替えて、%s となりました。\n", str_comma_population);
78
79 return 0;
80 }
```

♣ 千支を求める

▼ eto3.c

```

1 #include <stdio.h> // 標準入出力関数
2 #include <stdlib.h> // exit関数
3 #include <string.h> // 文字列操作関数
4
5 int main(int argc, char const *argv[]) {
6     // 入力を促すメッセージ表示
7     printf("何年生まれですか? (例: S30) > ");
8
9     // 8文字までキーボードから入力を受け取れるよう、バッファを用意
10    char buffer[8];
11
12    // キーボードから一行読み込む
13    // scanfはいろいろ支障があるので、
14    // fgets関数で、bufferへ、sizeof(buffer)-1文字分(7文字分),
15    // stdin(=キーボード)から読み込む
16    if (fgets(buffer, sizeof(buffer), stdin) == NULL) {
17        // エラーメッセージ出力
18        printf("キーボードから読み込めませんでした。\n");
19        exit(1);
20    }
21
22    // 改行文字が含まれているかどうか？
23    if (strchr(buffer, '\n') != NULL) {
24        // S30エンター のように4文字タイプされたときは、
25        // 改行文字(エンター)を文字列の終端記号に置換する
26        buffer[strlen(buffer) - 1] = '\0';
27    } else {
```

```
28 // buffer内に、改行文字が含まれていない場合（=8文字以上続けてタイプされた場合）  
29 // 最初の7文字は読み込まれているので、残りの入力ストリーム（キーバッファ）をクリアする  
30 while (getchar() != '\n')  
31 ;  
32 }  
33  
34 // 読み込んでいるか、確認  
35 // printf("キーボードからの入力は、\n%s です。\n", buffer);  
36  
37 // 入力文字列の先頭が、M, T, S, H, R で始まっているか確認し、西暦年に変換する  
38 int year; // 西暦年  
39 char era_initial; // 明治: 'M', 大正: 'T', 昭和: 'S', 平成: 'H', 令和: 'R'  
40 char era_year[3]; // 元号で何年か  
41  
42 // S30 と入力されていた場合、  
43 // S をera_initialにセットする  
44 era_initial = buffer[0];  
45 // 30 をera_yearにセットする  
46 era_year[0] = buffer[1];  
47 era_year[1] = buffer[2];  
48 era_year[2] = '\0';  
49  
50 char *endptr; // 数値に変換出来なかった文字  
51 // atoi関数は、文字列を数値に変換出来なかった場合にでも、0を返します。  
52 // 本当に、「0」という文字列が、0という数値に変換されたのか、  
53 // 区別が出来ないので、strtol関数を使います。  
54 year = strtol(era_year, &endptr, 10); // 10進数として変換する  
55  
56 // 数値変換不可の文字列の長さを調べる。  
57 if (strlen(endptr) != 0) {  
58 // エラーメッセージ出力  
59 printf("年数の入力が誤っています。\n", endptr);  
60 // プログラムを終了  
61 exit(1); // エラーコード1(異常終了)を返します  
62 }  
63  
64 switch (era_initial) {  
65 case 'M':  
66     year += 1867;  
67     break;  
68 case 'T':  
69     year += 1911;  
70     break;  
71 case 'S':  
72     year += 1925;  
73     break;  
74 case 'H':  
75     year += 1988;  
76     break;  
77 case 'R':  
78     year += 2018;
```

```

79     break;
80 default:
81     // エラーメッセージ出力
82     printf("元号の文字ではありません。\\n");
83     // プログラムを終了
84     exit(2); // エラーコード2(異常終了)を返します
85 }
86
87 // 干支の算出方法は、前回と同様
88 // 干支の配列(12で割り切れる年は申年)
89 char *eto[] = {"申", "酉", "戌", "亥", "子", "丑",
90                 "寅", "卯", "辰", "巳", "午", "未"};
91 printf("%c%s年(%d年)生まれのあなたの干支は、%s です。\\n",
92        era_initial, era_year, year, eto[year % 12]);
93
94 return 0;
95 }
```

♣ 生きてきた日数

▼ lifetime.c

```

1 ****
2 * 1980年1月1日生まれの人は、今日までに何日生きたことになるでしょうか？
3 * (生年月日は 19800101 と入力されます。)
4 * 昭和20年8月10日生まれの方は、どうでしょうか？
5 * (生年月日は 3200810 と入力されます。)
6 * 生後30000日目を迎えるのは、何年何月何日でしょうか？
7 ****
8
9 #include <stdio.h> // 標準入出力関数
10 #include <math.h> // 数学関数
11 #include <stdlib.h> // exit関数
12 #include <string.h> // 文字列操作関数
13 #include <time.h> // 日付時刻の操作用関数
14
15 // グレゴリオ暦から修正ユリウス日を返す為の関数
16 int gregorian_to_mjd(int year, int month, int date);
17
18 // 修正ユリウス日(mjd)からグレゴリオ暦(y, m, d)を返す為の関数
19 void mjd_to_gregorian(int mjd, int *y, int *m, int *d);
20
21 int main(int argc, char const *argv[]) {
22     // コマンドラインからの引数がなければ、使い方表示
23     if (argc == 1) {
24         printf("【使い方】\\n");
25         printf("今日まで何日生きたかを表示します。\\n");
26         printf("%s 19800101\\n", argv[0]);
27         printf("昭和20年8月10日生まれであれば、\\n");
28         printf("%s 3200810\\n", argv[0]);
29         printf("の様に入力して下さい。\\n");
30         exit(1);
31     }
```

```
32 }
33
34 int birthday; // 生年月日
35 char *endptr; // 数値に変換出来なかった文字
36 int b_year; // 誕生日のグレゴリオ暦年
37 int b_month; // 誕生日の月
38 int b_day; // 誕生日の日
39
40 // 簡易エラーチェック
41 // 数値変換不可の文字列の長さを調べる。
42 birthday = strtol(argv[1], &endptr, 10); // 10進数として変換する
43 if (strlen(endptr) != 0) {
44     // エラーメッセージ出力
45     printf("%s は、生年月日として認識出来ませんでした。\\n", endptr);
46     exit(1);
47 }
48
49 // 生年月日に分離する
50 // 年
51 // 元号入力なら、グレゴリオ暦へ変換
52 if (birthday < 1e7) {
53     int ge = birthday / 10000; // 上3桁 元号符号+元号年
54     switch (ge / 100) {
55         case 1:
56             b_year = ge % 100 + 1867;
57             break;
58         case 2:
59             b_year = ge % 100 + 1911;
60             break;
61         case 3:
62             b_year = ge % 100 + 1925;
63             break;
64         case 4:
65             b_year = ge % 100 + 1988;
66             break;
67         case 5:
68             b_year = ge % 100 + 2018;
69             break;
70     }
71 } else {
72     b_year = birthday / 10000; // 上4桁はグレゴリオ暦年
73 }
74 // 月
75 b_month = birthday / 100; // 下二桁を切り捨てて
76 b_month = b_month % 100; // 残った数の下2桁
77
78 // 日
79 b_day = birthday % 100; // 下2桁は日
80
81 // http://simd.jugem.jp/?eid=149 より引用
82 // 今日の日付を取得する
83 time_t now;
84 struct tm *ltm;
```

```

85 time(&now);
86 ltm = localtime(&now);
87
88 // 確認用
89 // printf( "%5d : [年]\n", ltm->tm_year + 1900 );
90 // printf( "%5d : [月]\n", ltm->tm_mon + 1 );
91 // printf( "%5d : [日]\n", ltm->tm_mday );
92 // printf( "%5d : [時]\n", ltm->tm_hour );
93 // printf( "%5d : [分]\n", ltm->tm_min );
94 // printf( "%5d : [秒]\n", ltm->tm_sec );
95 // printf( "%5d : [曜日]\n", ltm->tm_wday );
96 // printf( "%5d : [経過日数]\n", ltm->tm_yday );
97 // printf( "%5d : [夏時間の有無]\n", ltm->tm_isdst );
98
99 int t_year = ltm->tm_year + 1900; // 今日のグレゴリオ暦年
100 int t_month = ltm->tm_mon + 1; // 今日の月
101 int t_day = ltm->tm_mday; // 今日の日
102
103 // strftime関数を用いて文字列にすることも可能
104 // char str_time[100];
105 // int maxsize = 100;
106 // char *format = "%Y年%m月%d日 %H:%M";
107 // strftime(str_time, maxsize, format, ltm);
108 // printf("%s\n", str_time);
109
110 // まともに暦の日付計算をしようとすると大変なので、
111 // ある基準日からの経過日数を求めることにします。
112 // 今日は、 基準日から何日目
113 // 誕生日は、基準日から何日目 と分かれば、
114 // 引き算することで、日数を計算出来ます。
115 // ユリウス通日（つうじつ）として知られており、
116 // 紀元前4713年1月1日 を第一日として、
117 // 天文学などの分野で用いられています。
118 // ユリウス通日は桁数が大きくなるので、
119 // ここから240万日を引いた修正ユリウス日も用いられます。
120 // https://ja.wikipedia.org/wiki/ユリウス通日
121 // に公式がありますので、プログラミングしました。
122 // https://ufcpp.net/study/algorithm/o\_days.html
123 // にも分かりやすい説明がありますので、ご一読ください。
124
125 printf("今日は      %4d 年 %2d 月 %2d 日です。\\n", t_year, t_month, t_day);
126 printf("誕生日は      %4d 年 %2d 月 %2d 日です。\\n", b_year, b_month, b_day);
127
128 // 今日の修正ユリウス日
129 int today_mjd = gregorian_to_mjd(t_year, t_month, t_day);
130 // printf("%5d\\n", today_mjd);
131 int birthday_mjd = gregorian_to_mjd(b_year, b_month, b_day);
132 // printf("%5d\\n", birthday_mjd);
133
134 // 結果表示
135 printf("今日は生まれて %5d 日目 です。\\n", today_mjd - birthday_mjd + 1);
136
137 // 生まれてから30000日後がいつかを求めます。

```

```

138 int life_30000_mjd = birthday_mjd + 30000;
139 int l_year, l_month, l_day;
140 // l_year, l_month, l_day に値がセットされるよう、アドレスを渡します。
141 mjd_to_gregorian(life_30000_mjd, &l_year, &l_month, &l_day);
142 printf("生後三万日は %4d 年 %2d 月 %2d 日です。\\n", l_year, l_month, l_day);
143
144 return 0;
145 }
146
147 // グレゴリオ暦から修正ユリウス日を返す為の関数
148 int gregorian_to_mjd(int year, int month, int date) {
149 // 1月や2月は前年の13月,14月として扱う
150 if (month == 1 || month == 2) {
151 month += 12;
152 year -= 1;
153 }
154
155 // 公式に従い、修正ユリウス日mjdを求める
156 int mjd = floor(365.25 * year)
157     + floor(year / 400)
158     - floor(year / 100)
159     + floor(30.59 * (month - 2))
160     + date
161     - 678912;
162
163 // 算出結果を返す
164 return mjd;
165 }
166
167 // 修正ユリウス日(mjd)からグレゴリオ暦(y, m, d)を返す為の関数
168 void mjd_to_gregorian(int mjd, int *y, int *m, int *d) {
169 // 複数の値を返すときは、ポインタで返します
170
171 int n = mjd + 678881;
172 // floor関数はdouble型を返すので、整数型(int)へキャスト
173 int a = 4*n + 3 + 4*(int)floor((3.0/4.0) * (4*(n+1)/146097 + 1));
174 int b = 5 * ((a % 1461) / 4) + 2;
175 *y = a / 1461;
176 *m = b / 153 + 3;
177 *d = (b % 153) / 5 + 1;
178 // 1月や2月は前年の13月,14月として扱う
179 if (*m == 13 || *m == 14) {
180 *m -= 12;
181 *y += 1;
182 }
183 }
```

♣ カレンダー

▼calendar.c

```

1 ****
2 * 年と月を入力すると、
```

```

3  * その月のカレンダーを表示するプログラムを作ってみましょう。
4  * 「ツェラーの公式」を用いると曜日を求めることが出来ます。
5  * ( グレゴリオ暦1年1月1日は月曜日ですので、
6  * 7で割った余りを求めてことで、曜日が計算出来ます )
7  ****
8
9 #include <stdio.h> // 標準入出力関数
10 #include <stdlib.h> // exit関数
11 #include <string.h> // 文字列操作関数
12
13 // ツェラーの公式(Zeller's congruence)により 曜日を返す関数
14 // 戻り値が 0, 1, 2, 3, 4, 5, 6 の場合、
15 // それぞれ 日曜日、月曜日、火曜日、水曜日、木曜日、金曜日、土曜日
16 int zeller(int year, int month, int day);
17
18 // 今月が何日まであるかを返す
19 int last_day_of_month(int year, int month);
20
21 int main(int argc, char const *argv[]) {
22     //
23     // コマンドラインからの引数がなければ、使い方表示
24     //
25     if (argc == 1) {
26         printf("【使い方】\n");
27         printf("カレンダーを表示します。\n");
28         printf("2030年 6月のカレンダーであれば\n");
29         printf("%s 2030 6\n", argv[0]);
30         printf("の様に入力して下さい。.\n");
31         exit(0); // プログラム終了
32     }
33
34     //
35     // 変数宣言
36     //
37     int year; // グレゴリオ暦年
38     int month; // 月
39     int day; // 日(カレンダー表示用)
40
41     char *err_str; // 数値に変換出来なかった文字
42
43     int day_of_week; // 曜日
44     int week_number; // 月の第何週か？
45     char *days_name[] = {"日", "月", "火", "水", // 曜日の名前
46                           "木", "金", "土"};
47
48     int last_day; // 今月は何日が最後の日か？
49
50     int h; // 今月1日が何曜日であるか？(ツェラーの公式)
51
52     //
53     // 簡易エラーチェック
54     //
55

```

```

56 // 数値変換不可の文字列の長さを調べる。
57 year = strtol(argv[1], &err_str, 10); // 10進数として変換する
58 if (strlen(err_str) != 0) {
59     // エラーメッセージ出力
60     printf("%s は、グレゴリオ暦年として認識出来ませんでした。\\n", err_str);
61     exit(1); // プログラム終了
62 }
63 // 数値変換不可の文字列の長さを調べる。
64 month = strtol(argv[2], &err_str, 10); // 10進数として変換する
65 if (strlen(err_str) != 0) {
66     // エラーメッセージ出力
67     printf("%s は、月として認識出来ませんでした。\\n", err_str);
68     exit(1); // プログラム終了
69 }
70
71 //
72 // カレンダー表示処理
73 //
74
75 // カレンダーの年月と曜日名を表示
76 printf("\\n    %4d 年 %2d 月\\n", year, month);
77 for (day_of_week = 0; day_of_week < 7; day_of_week++) {
78     printf(" %s", days_name[day_of_week]);
79 }
80 printf("\\n");
81
82 // ツェラーの公式で、その月の 1 日が何曜日始まりであるか、取得する。
83 // 戻り値が 0, 1, 2, 3, 4, 5, 6 の場合、
84 // それぞれ 日曜日、月曜日、火曜日、水曜日、木曜日、金曜日、土曜日
85 h = zeller(year, month, 1);
86
87 // 今月の最終日を求める
88 last_day = last_day_of_month(year, month);
89
90 // 水曜日始まりの月の場合、
91 // 日、月、火 の欄に日付を表示させたくないでの、
92 // day = 1 - h からスタートする
93 day        = 1 - h;
94 day_of_week = 0;
95 do {
96     if (day < 1) {
97         printf("    "); // 1日以前なら空欄を出力
98         day++;
99     } else {
100        printf(" %2d", day);
101        day++;
102    }
103    day_of_week++;
104    if (day_of_week % 7 == 0) {
105        printf("\\n"); // 週送り
106    }
107 } while (day <= last_day);
108 printf("\\n");

```

```

109     return 0;
110 }
111
112
113 // ツェラーの公式(Zeller's congruence)により 曜日を返す関数
114 // 戻り値が 0, 1, 2, 3, 4, 5, 6 の場合、
115 // それぞれ 日曜日、月曜日、火曜日、水曜日、木曜日、金曜日、土曜日
116 int zeller(int y, int m, int d) {
117     // 1月、2月は前年の13月、14月として計算
118     if (m == 1 || m == 2) {
119         y -= 1;
120         m += 12;
121     }
122
123     // 曜日の算出
124     return (y + y / 4 - y / 100 + y / 400 + (13 * m + 8) / 5 + d) % 7;
125 }
126
127 // 今月が何日まであるかを返す
128 // int last_day[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}
129 // のように配列に値を持たせても良い。
130 int last_day_of_month(int year, int month) {
131     switch (month) {
132     case 2:
133         // 閏年判定
134         if ((year % 4 == 0 && year % 100 != 0) || year % 400 == 0) {
135             return 29;
136         } else {
137             return 28;
138         }
139     // 小の月
140     case 4:
141     case 6:
142     case 9:
143     case 11:
144         return 30;
145     // 大の月
146     case 1:
147     case 3:
148     case 5:
149     case 7:
150     case 8:
151     case 10:
152     case 12:
153         return 31;
154     }
155 }
```

♣ 漢数字

▼ num2kan.c

```

1  ****
2  * 算用数字 302 を 漢数字にします。
3  * 無量大数までの変換が出来ます。
4  * 参考: https://ja.wikipedia.org/wiki/命數法
5  *
6  * 正規表現については、
7  * https://ja.wikipedia.org/wiki/正規表現
8  * https://rubular.com 参照
9  ****
10 #include <stdio.h> // 標準入出力関数
11 #include <stdlib.h> // exit関数
12 #include <string.h> // 文字列操作関数
13 #include <regex.h> // 正規表現
14
15 // 漢字変換に用いる定数
16 // main の外側に書かれているので、どの関数からでも参照可能
17 // const は 変更禁止の為の修飾子
18 const char *digit[] = {"〇", "一", "二", "三", "四",
19                         "五", "六", "七", "八", "九"};
20 const char *unit1[] = {"千", "百", "十", ""};
21 const char *unit2[] = {"", "万", "億", "兆", "京",
22                       "垓", "秭", "穰", "溝", "澗",
23                       "正", "載", "極", "恒河沙", "阿僧祇",
24                       "那由他", "不可思議", "無量大数"};
25
26 // 4桁の漢数字への変換を行う関数
27 char *num2kan(char *str);
28
29 int main(int argc, char const *argv[]) {
30
31     // コマンドラインからの引数がなければ、使い方を表示
32     if (argc == 1) {
33         printf("【使い方】\n");
34         printf(
35             "算用数字を漢数字にするプログラムです。無量大数まで変換出来ます。\n");
36         printf("%s 1234567890\n", argv[0]);
37         printf("の様に入力して下さい。\n");
38         printf("十二億三千四百五十六万七千八百九十\n");
39         printf("を表示します。\n");
40         exit(0);
41     }
42
43     // コマンドライン引数チェック
44     // '0'-'9' 以外の文字が含まれていたら、エラー表示し、終了する。
45     // ( 正規表現を用いたが、一桁ずつ読み込み、「0'-9」か、チェックしても良い )
46     // 正規表現のC言語でのコーディングについては、
47     // C 言語での正規表現 https://www.delftstack.com/ja/howto/c/c-regex/
48     // が参考になります。
49
50     char *reg = "[0-9]+"; // 全て数字であるか判定するための正規表現
51     regex_t regst;
52     // 正規表現のコンパイル

```

```

53     if (regcomp(&regist, reg, REG_EXTENDED)) {
54         return 1;
55     }
56     // argv[1]が、正規表現にマッチするか、実行
57     if (regexec(&regist, argv[1], 0, NULL, 0)) { // マッチしない場合
58         printf("%s は 算用数字として認識出来ませんでした。\\n", argv[1]);
59         exit(1);
60     }
61     // コンパイル結果の開放は必須
62     regfree(&regist);
63
64     // 最大取り扱い桁数 9999無量大数(10^68)までの数
65     if (strlen(argv[1]) > 4 + 68 + 1) {
66         printf("%s は 長すぎます。73桁までの数字を入力して下さい。\\n", argv[1]);
67         exit(1);
68     }
69
70     // 変数宣言
71     int digit_length; // 何桁の数字か
72     int i, u;
73     char string_number[4 + 68 + 1 + 1]; // 算用数字 最大取り扱い桁数
74                                         // 9999無量大数(9999*1e68)まで
75     strcpy(string_number, "");           // 初期化
76     // 変換後の漢数字(4桁毎に7文字21バイトになるので) また万億兆などの文字数も追加
77     char chinese_numeral[(7 * 3) * (72 / 4) + sizeof(unit2) + 1];
78     strcpy(chinese_numeral, "");
79     char work[7 * 3 + 1];
80     strcpy(work, "");
81
82     // 漢数字への変換処理
83     // 4桁ごとに区切って変換できるよう、先頭に0を付与する
84     // 12 3456 7890 => 0012 3456 7890
85     digit_length = strlen(argv[1]);      // 何桁の数であるか？
86     int giving_zero = 4 - digit_length % 4; // 先頭に付与すべき0の数
87     giving_zero %= 4;                      // 4 なら 0 にする
88     for (i = 0; i < giving_zero; i++) {
89         strcat(string_number, "0");
90     }
91     strcat(string_number, argv[1]);
92     // 何桁の数字か
93     digit_length = strlen(string_number);
94
95     // 4桁毎に漢数字に変換
96     // 4桁までなら unit2 ""
97     // 8桁までなら unit2 "万"
98     // 12桁までなら unit2 "億"
99     // 等を付与する
100    int unit2_index = (digit_length / 4) - 1;
101    for (u = 0; u < digit_length / 4; u++, unit2_index--) {
102        // string_number から 各ユニット毎に、4文字ずつ work にコピー
103        work[0] = string_number[4 * u + 0];
104        work[1] = string_number[4 * u + 1];
105        work[2] = string_number[4 * u + 2];

```

```

106 work[3] = string_number[4 * u + 3];
107 work[4] = '\n';
108
109 // 4桁毎に漢数字に変換、結果を連結する
110 strcat(chinese_numeral, num2kan(work));
111 // "億", "万", "" の付与
112 strcat(chinese_numeral, unit2[unit2_index]);
113 }
114
115 // 0 の場合のみ、空文字列のままなので、"○" にする。
116 if (strlen(chinese_numeral) == 0) {
117     strcpy(chinese_numeral, digit[0]);
118 }
119
120 // 結果表示
121 printf("%s\n", chinese_numeral);
122
123 return 0;
124 }

// 4桁 の 漢数字へ変換する関数
126 char *num2kan(char *str) {
127     // 変数宣言
128     int i;
129     char work[7 * 3 + 1]; // 三千四百五十六 7文字*3バイト+終端文字1バイト
130     strcpy(work, ""); // 初期化
131
132     // 一つずつ漢数字に変換する
133     for (i = 0; i < 4; i++) {
134         switch (str[i]) {
135             case '0':
136                 break; // 何もせず、次の桁へ進む
137             case '1':
138                 // 一の位のみ "一"と書く
139                 if (i == 3) {
140                     strcat(work, digit[str[i] - '0']); // str[i]-'0' 文字0から数値0へ変換
141                 }
142                 strcat(work, unit1[i]); // "千", "百", "十", "" のいずれか
143                 break;
144             default:
145                 strcat(work, digit[str[i] - '0']); // "一" ~ "九"
146                 strcat(work, unit1[i]); // "千", "百", "十", "" のいずれか
147                 break;
148         }
149     }
150 }
151
152 // 変換結果を返す
153 return strcpy(str, work);
154 }

```

♣ 英単語帳アプリ

▼ word_book.c

```

1  ****
2  * 英単語帳アプリを創ってみましょう。
3  * I -> わたし
4  * love -> 愛する
5  * you -> あなた
6  * と答えられたら、満点です。
7  * 10問出題して、英語力向上を目指しましょう。
8  ****
9
10 #include "mt.h"      // Mersenne Twister法による乱数
11 #include <stdio.h>    // 標準入出力関数
12
13 #define CHOICES_SIZE   3 // 三択で出題する
14 #define REGISTERD_WORD 10 // 登録単語数
15 #define TRUE            1 // 真
16 #define FALSE           0 // 偽
17
18 int main(int argc, char const *argv[]) {
19     // 出題用配列
20     char *english_words[] = {"", "I", "love", "you",
21                             "C", "language", "lesson", "happy",
22                             "hacker", "programming", "computer"};
23     char *japanese_words[] = {"", "わたし", "愛する", "あなた",
24                             "C", "言語", "学習", "幸せ",
25                             "技術者", "プログラミング", "コンピュータ"};
26
27     // 三択で出題する
28     int choices[CHOICES_SIZE + 1]; // 0 は未使用とするため、+1
29     int candidate;                // 選択肢の候補
30     int question_number;          // 第何問目か？
31     int correct;                 // 正答
32     int answer;                  // 入力された回答
33     int score;                   // 得点
34     int flag;                    // フラグ
35     int i;
36
37     // オープニング
38     printf("*****\n");
39     printf("*\n");
40     printf("*      英単語ゲームへようこそ\n");
41     printf("*\n");
42     printf("*****\n");
43     printf("\n");
44
45     // 出題処理
46
47     // 10題出題する
48     score = 0;
49     for (question_number = 1; question_number <= 10; question_number++) {
50         printf("【第 %d 問】\n", question_number);

```

```

51
52 // 選択肢の初期化
53 for (i = 0; i <= CHOICES_SIZE; i++) {
54     choices[i] = 0;
55 }
56
57 // choices[1], [2], [3] に単語番号(添字)をセット
58 do {
59     // 1から英単語数(REGISTERD_WORD)までの乱数を取得
60     candidate = genrand_int32() % REGISTERD_WORD + 1;
61     // このcandidate(候補)が、choices(選択肢)に登録済みか調べる。
62     int flag = FALSE;
63     for (i = 1; i <= CHOICES_SIZE; i++) {
64         if (candidate == choices[i]) {
65             flag = TRUE; // すでに選ばれていた
66             break;
67         }
68     }
69     // 選択肢には未登録だったので、登録する
70     if (flag == FALSE) {
71         for (i = 1; i <= CHOICES_SIZE; i++) {
72             if (choices[i] == 0) {
73                 // i番目の選択肢として登録する
74                 choices[i] = candidate;
75                 break;
76             }
77         }
78     }
79     // choice[3] (最後の選択肢) に0以外の値がセットされるまで繰り返す
80 } while (choices[CHOICES_SIZE] == 0);
81
82 // choice[1], [2], [3] の中から、いずれかを正解として設定する
83 correct = genrand_int32() % 3 + 1;
84
85 // correct を正解として設定したので、
86 // choices[correct] の単語を出題する
87 printf("%s\n", english_words[choices[correct]]);
88
89 // 選択肢を提示する
90 // (choices[correct] の単語が必ず選択肢に含まれている)
91 for (i = 1; i <= CHOICES_SIZE; i++) {
92     printf("%d: %s ", i, japanese_words[choices[i]]);
93 }
94 printf("\n");
95
96 // 1 ~ 3 までの入力を求める
97 do {
98     scanf("%d", &answer);
99     while (getchar() != '\n')
100         ; // キーバッファ読み飛ばす
101 } while (answer <= 0 || answer > CHOICES_SIZE);
102
103 // 正解判定

```

```

104     if (answer == correct) {
105         printf("正解です！\n\n");
106         score++; // 得点加算
107     } else {
108         printf("残念。正解は、%d: %s です。\n\n", correct,
109                japanese_words[choices[correct]]);
110     }
111 }
112
113 // 総合結果発表
114 printf("*****\n");
115 printf("*          結果発表          *\n");
116 printf("*          10 問中 %d 問 正解\n", score);
117 printf("*          おめでとうございます！\n");
118 printf("*****\n");
119 printf("\n");
120
121 return 0;
122 }
123
124
125
126 }
```

♣ 計算ゲーム

▼ calculation_game3.c

```

1 ****
2 * バージョンアップ版です。
3 * より片寄りのない乱数を生成する為、
4 * 数学者の松本真さんと西村拓士さんによって考案された
5 * メルセンヌツイスター (Mersenne Twister) 法を用いています。
6 * また、ゲームらしく、10題解くのにかかった時間も表示しています。
7 ****
8 #include "mt.h" // Mersenne Twister法による乱数
9 // 以下の関数が用意されている。
10 // genrand_int32() // 符号なし32ビット長整数
11 // genrand_int31() // 符号なし31ビット長整数
12 // genrand_real1() // 一様実乱数[0,1] (32ビット精度)
13 // genrand_real2() // 一様実乱数[0,1] (32ビット精度)
14 // genrand_real3() // 一様実乱数(0,1) (32ビット精度)
15 // genrand_res53() // 一様実乱数[0,1] (53ビット精度)
16 // AからBの範囲の整数の乱数が欲しいときには
17 // genrand_int32()%(B-A+1)+A;
18 // のような関数を用いればよい。
19
20 #include <stdio.h> // 標準入出力関数
21 #include <math.h> // 数学関数
22 #include <string.h> // 文字列操作関数
23 #include <time.h> // time関数
24
25 int main(int argc, char const *argv[]) {
```

```

26 int operand1;           // 第一被演算子
27 int operand2;           // 第二被演算子
28 char operator;          // 演算子 ( '+', '-', '*', '/' )
29 int digit;              // 桁数
30 int result;             // 演算結果
31 int your_answer;        // 回答
32 int correct;            // 正答数
33 time_t start_time;      // ゲーム開始時刻
34 time_t finish_time;     // ゲーム完了時刻
35 int n;                  // 繰り返し回数
36
37 // オープニング
38 printf("*****\n");
39 printf("*\n");
40 printf("*      計算ゲームへようこそ\n");
41 printf("*\n");
42 printf("*****\n");
43 printf("\n");
44
45 // 桁数をリクエスト
46 do {
47     printf("何桁の数字で挑戦しますか？ \n");
48     printf("一桁: 1 二桁: 2 三桁: 3 四桁: 4 \n");
49     scanf("%d", &digit);
50     while (getchar() != '\n')
51         ; // キーバッファ読み飛ばす
52 } while (digit <= 0 || digit >= 5);
53
54 // 演算子をリクエスト
55 do {
56     printf("どの計算に挑戦しますか？ \n");
57     printf("加算: + 減算: - 乗算: * 除算: /\n");
58     scanf("%c", &operator);
59     while (getchar() != '\n')
60         ; // キーバッファ読み飛ばす
61 } while (operator!= '+' &&
62       operator!= '-' &&
63       operator!= '*' &&
64       operator!= '/');
65
66 // 出題処理
67 correct = 0; // 正答数初期化
68 time(&start_time); // ゲーム開始時刻を取得
69 for (n = 1; n <= 10; n++) {
70     // digit桁の乱数を求める
71     // genrand_int32 は 32bit(0-約43億) の整数型の乱数を返すので、
72     // 10(または100, 1000, 10000)で割った余りが得たい乱数となる。
73     // (int) は、キャストと呼ばれる。
74     // pow関数の戻り値はdouble型なので、int型へ変換している。
75     if (operator!= '/') {
76         operand1 = genrand_int32() % (int)pow(10, digit);
77     } else {
78         // 除算時、同じ桁同士で演算すると、ほぼ0となるので、調整。

```

```

79     operand1 = genrand_int32() % (int)pow(10, digit + 1);
80 }
81 do {
82     operand2 = genrand_int32() % (int)pow(10, digit);
83 } while (operator == '/' && operand2 == 0); // 0 の除算は定義されていない
84
85 // 正答を用意
86 switch (operator) {
87 case '+':
88     result = operand1 + operand2;
89     break;
90 case '-':
91     result = operand1 - operand2;
92     break;
93 case '*':
94     result = operand1 * operand2;
95     break;
96 case '/':
97     result = operand1 / operand2;
98     break;
99 }
100
101 // 何回目のゲームか、表示
102 switch (operator) {
103 case '+':
104     printf("足し算ゲーム %d 回目\n", n);
105     break;
106 case '-':
107     printf("引き算ゲーム %d 回目\n", n);
108     break;
109 case '*':
110     printf("掛け算ゲーム %d 回目\n", n);
111     break;
112 case '/':
113     printf("割り算ゲーム %d 回目\n", n);
114     break;
115 }
116
117 // 出題する
118 printf("%d %c %d = ?\n", operand1, operator, operand2);
119
120 // 回答を受け取る
121 scanf("%d", &your_answer);
122 while (getchar() != '\n')
123     ; // キーバッファ読み飛ばす
124
125 // 正解発表と正答数のカウント
126 if (your_answer == result) {
127     printf("正解です。\\n");
128     correct++;
129 } else {
130     printf("正解は、%d です。\\n", result);
131 }
```

```

132 }
133 // ゲーム完了時刻を取得
134 time(&finish_time);
135
136 // 総合結果発表
137 printf("*****\n");
138 printf("*          結果発表\n");
139 printf("*          10問中 %d 問 正解\n", correct);
140 printf("*          %ld 秒 でクリア !\n", finish_time - start_time);
141 printf("*          おめでとうございます !\n");
142 printf("\n");
143 printf("*****\n");
144 printf("\n");
145 printf("\n");
146 printf("*****\n");
147 printf("\n");
148
149 return 0;
150
151 // 1 + 1 などが出現せぬよう、
152 // 被演算子が重複しないようにしましたが、
153 // 重複を許可する、
154 // 引き算の結果は正の範囲に納める、
155 // 剰余演算を追加するなど、
156 // いろいろ発展させていって下さい。
157 }

```

♣ 数当てゲーム

▼match_number.c

```

1 ****
2 * 数当てゲーム Match Number
3 *
4 * 事前に用意された3桁の数字を、ヒントをもとに当てていくゲームです。
5 * 用意された数字が 9 4 3 だとします。
6 * 9 9 9と入れると、9は正解ですので、1つ正解と表示します。
7 * 3 6 9と入れると、3と9は正解ですので、2つ正解と表示します。
8 * 4 3 9と入れると、(順番は違いますが) 3つとも合っていますので、
9 * 3つ正解と表示します。
10 * これを繰り返すことで、事前に用意された数字、9 4 3を当てるゲームです。
11 ****
12
13 #include <stdio.h> // 標準入出力関数
14 #include <string.h> // 文字列操作関数
15
16 int main(int argc, char const *argv[]) {
17     int right_number[] = {9, 4, 3}; // 事前に用意された数
18                                         // 993など重複した数は対象外
19     char your_number[8];           // 入力された数字
20     int number;
21     int unique_number[3];         // 重複を除いた数
22                                         // 999の入力なら 9

```

```

23                                     // 990 の入力なら 9, 0
24                                     // 987 の入力なら 9, 8, 7 が入る
25 int score;                         // いくつ合っていたか
26 int i, j;                          // loop counter
27
28 while(1) {
29     printf("三桁の数字を入力して下さい (終了:quit)\n");
30     scanf("%s", your_number);
31     while (getchar() != '\n')
32         ;
33
34     // "quit" が入力されたらプログラム終了
35     if (strcmp(your_number, "quit") == 0) {
36         break;
37     }
38
39     // 入力された文字'n'を数nに変換
40     for (i = 0; i < 3; i++) {
41         unique_number[i] = your_number[i] - '0';
42     }
43
44     // 重複を排除する
45     // (990と入力されたならば、90 をMatch Number の対象にする)
46     for (i = 2; i >= 1; i--) {    // un[2]は、un[1], un[0]と等しいか？
47                                     // un[1]は、un[0]と等しいか？
48         for (j = i - 1; j >= 0; j--) {
49             if (unique_number[i] == unique_number[j]) {
50                 unique_number[i] = -1; // 自分より小さい添字の要素が、
51                                     // 自分自身と重複していることが判明したため、
52                                     // 未使用の意味で、-1をセットする
53             }
54         }
55     }
56
57     // いくつ合っているか、出力
58     score = 0;
59     for (i = 0; i < 3; i++) {
60         if (unique_number[i] != -1) {
61             number = unique_number[i];
62         } else {
63             break;
64         }
65         for (j = 0; j < 3; j++) {
66             if (number == right_number[j]) {
67                 score++;
68                 break;
69             }
70         }
71     }
72     printf("正解数: %d \n\n", score);
73
74     // 入力された数値の配列と、正解の配列が等しいことを確認する
75     if (unique_number[0] == right_number[0] &&

```

```

76     unique_number[1] == right_number[1] &&
77     unique_number[2] == right_number[2]) {
78     // 代えて次のように書くことも出来ます。
79     // if(memcmp(unique_number, right_number, sizeof(unique_number)) == 0){
80     printf("おめでとうございます。正解です！！\n");
81     break;
82   }
83 } // while end
84 }
```

♣ ビットマップフォント

▼ bitmap_font.c

```

1 #include <stdio.h>
2
3 // 数を表示するための関数
4 void show_font(char *number) {
5   int row;      // 行
6   int data;
7   int column;  // 列
8   int mask;
9
10  for (row = 0; row < 5; row++) {
11    // 一行目のデータを取り出す
12    data = number[row];
13
14    for (column = 0; column < 4; column++) {
15      // 覆い隠すためのマスクを用意する
16      mask = 0x08;
17      // column ビット 左シフトする
18      // (mask = 0b1000 -> 0b0100 -> 0b0010 -> 0b0001 と変化していく)
19      mask = mask >> column;
20      // mask と ビット論理積(&)を取ると、
21      // data の 3bit目が 0 か 1 かを 判定できる
22      // printf("確認: %x & %x = %x\n", data, mask, data & mask);
23      if ((data & mask) != 0){
24        printf("■");
25      } else {
26        printf("□");
27      }
28    }
29    printf("\n");
30  }
31
32
33  int main(int argc, char const *argv[]) {
34
35    // ビットマップフォントの定義
36    // char型の配列を用意し、
37    // 0-9まで、それぞれの数字の形になるよう、
38    // 十六進数でビットを立てます。
39    char bitmap[10][5] ={{0x0f,  // 0b1111 // 0
40                        0x00,  // 0b0000 // 0
41                        0x00,  // 0b0000 // 0
42                        0x00,  // 0b0000 // 0
43                        0x00}, // 0b0000 // 0
44                        {0x00,  // 0b0000 // 0
45                        0x00,  // 0b0000 // 0
46                        0x00,  // 0b0000 // 0
47                        0x00,  // 0b0000 // 0
48                        0x00}, // 0b0000 // 0
49                        {0x00,  // 0b0000 // 0
50                        0x00,  // 0b0000 // 0
51                        0x00,  // 0b0000 // 0
52                        0x00,  // 0b0000 // 0
53                        0x00}, // 0b0000 // 0
54                        {0x00,  // 0b0000 // 0
55                        0x00,  // 0b0000 // 0
56                        0x00,  // 0b0000 // 0
57                        0x00,  // 0b0000 // 0
58                        0x00}, // 0b0000 // 0
59                        {0x00,  // 0b0000 // 0
60                        0x00,  // 0b0000 // 0
61                        0x00,  // 0b0000 // 0
62                        0x00,  // 0b0000 // 0
63                        0x00}, // 0b0000 // 0
64                        {0x00,  // 0b0000 // 0
65                        0x00,  // 0b0000 // 0
66                        0x00,  // 0b0000 // 0
67                        0x00,  // 0b0000 // 0
68                        0x00}, // 0b0000 // 0
69                        {0x00,  // 0b0000 // 0
70                        0x00,  // 0b0000 // 0
71                        0x00,  // 0b0000 // 0
72                        0x00,  // 0b0000 // 0
73                        0x00}, // 0b0000 // 0
74                        {0x00,  // 0b0000 // 0
75                        0x00,  // 0b0000 // 0
76                        0x00,  // 0b0000 // 0
77                        0x00,  // 0b0000 // 0
78                        0x00}, // 0b0000 // 0
79                        {0x00,  // 0b0000 // 0
80                        0x00,  // 0b0000 // 0
81                        0x00,  // 0b0000 // 0
82                        0x00,  // 0b0000 // 0
83                        0x00}, // 0b0000 // 0
84                        {0x00,  // 0b0000 // 0
85                        0x00,  // 0b0000 // 0
86                        0x00,  // 0b0000 // 0
87                        0x00,  // 0b0000 // 0
88                        0x00}, // 0b0000 // 0
89                        {0x00,  // 0b0000 // 0
90                        0x00,  // 0b0000 // 0
91                        0x00,  // 0b0000 // 0
92                        0x00,  // 0b0000 // 0
93                        0x00}, // 0b0000 // 0
94                        {0x00,  // 0b0000 // 0
95                        0x00,  // 0b0000 // 0
96                        0x00,  // 0b0000 // 0
97                        0x00,  // 0b0000 // 0
98                        0x00}, // 0b0000 // 0
99                        {0x00,  // 0b0000 // 0
100                       0x00,  // 0b0000 // 0
101                      0x00,  // 0b0000 // 0
102                      0x00,  // 0b0000 // 0
103                      0x00}, // 0b0000 // 0
104                      {0x00,  // 0b0000 // 0
105                      0x00,  // 0b0000 // 0
106                      0x00,  // 0b0000 // 0
107                      0x00,  // 0b0000 // 0
108                      0x00}, // 0b0000 // 0
109                      {0x00,  // 0b0000 // 0
110                      0x00,  // 0b0000 // 0
111                      0x00,  // 0b0000 // 0
112                      0x00,  // 0b0000 // 0
113                      0x00}, // 0b0000 // 0
114                      {0x00,  // 0b0000 // 0
115                      0x00,  // 0b0000 // 0
116                      0x00,  // 0b0000 // 0
117                      0x00,  // 0b0000 // 0
118                      0x00}, // 0b0000 // 0
119                      {0x00,  // 0b0000 // 0
120                      0x00,  // 0b0000 // 0
121                      0x00,  // 0b0000 // 0
122                      0x00,  // 0b0000 // 0
123                      0x00}, // 0b0000 // 0
124                      {0x00,  // 0b0000 // 0
125                      0x00,  // 0b0000 // 0
126                      0x00,  // 0b0000 // 0
127                      0x00,  // 0b0000 // 0
128                      0x00}, // 0b0000 // 0
129                      {0x00,  // 0b0000 // 0
130                      0x00,  // 0b0000 // 0
131                      0x00,  // 0b0000 // 0
132                      0x00,  // 0b0000 // 0
133                      0x00}, // 0b0000 // 0
134                      {0x00,  // 0b0000 // 0
135                      0x00,  // 0b0000 // 0
136                      0x00,  // 0b0000 // 0
137                      0x00,  // 0b0000 // 0
138                      0x00}, // 0b0000 // 0
139                      {0x00,  // 0b0000 // 0
140                      0x00,  // 0b0000 // 0
141                      0x00,  // 0b0000 // 0
142                      0x00,  // 0b0000 // 0
143                      0x00}, // 0b0000 // 0
144                      {0x00,  // 0b0000 // 0
145                      0x00,  // 0b0000 // 0
146                      0x00,  // 0b0000 // 0
147                      0x00,  // 0b0000 // 0
148                      0x00}, // 0b0000 // 0
149                      {0x00,  // 0b0000 // 0
150                      0x00,  // 0b0000 // 0
151                      0x00,  // 0b0000 // 0
152                      0x00,  // 0b0000 // 0
153                      0x00}, // 0b0000 // 0
154                      {0x00,  // 0b0000 // 0
155                      0x00,  // 0b0000 // 0
156                      0x00,  // 0b0000 // 0
157                      0x00,  // 0b0000 // 0
158                      0x00}, // 0b0000 // 0
159                      {0x00,  // 0b0000 // 0
160                      0x00,  // 0b0000 // 0
161                      0x00,  // 0b0000 // 0
162                      0x00,  // 0b0000 // 0
163                      0x00}, // 0b0000 // 0
164                      {0x00,  // 0b0000 // 0
165                      0x00,  // 0b0000 // 0
166                      0x00,  // 0b0000 // 0
167                      0x00,  // 0b0000 // 0
168                      0x00}, // 0b0000 // 0
169                      {0x00,  // 0b0000 // 0
170                      0x00,  // 0b0000 // 0
171                      0x00,  // 0b0000 // 0
172                      0x00,  // 0b0000 // 0
173                      0x00}, // 0b0000 // 0
174                      {0x00,  // 0b0000 // 0
175                      0x00,  // 0b0000 // 0
176                      0x00,  // 0b0000 // 0
177                      0x00,  // 0b0000 // 0
178                      0x00}, // 0b0000 // 0
179                      {0x00,  // 0b0000 // 0
180                      0x00,  // 0b0000 // 0
181                      0x00,  // 0b0000 // 0
182                      0x00,  // 0b0000 // 0
183                      0x00}, // 0b0000 // 0
184                      {0x00,  // 0b0000 // 0
185                      0x00,  // 0b0000 // 0
186                      0x00,  // 0b0000 // 0
187                      0x00,  // 0b0000 // 0
188                      0x00}, // 0b0000 // 0
189                      {0x00,  // 0b0000 // 0
190                      0x00,  // 0b0000 // 0
191                      0x00,  // 0b0000 // 0
192                      0x00,  // 0b0000 // 0
193                      0x00}, // 0b0000 // 0
194                      {0x00,  // 0b0000 // 0
195                      0x00,  // 0b0000 // 0
196                      0x00,  // 0b0000 // 0
197                      0x00,  // 0b0000 // 0
198                      0x00}, // 0b0000 // 0
199                      {0x00,  // 0b0000 // 0
200                      0x00,  // 0b0000 // 0
201                      0x00,  // 0b0000 // 0
202                      0x00,  // 0b0000 // 0
203                      0x00}, // 0b0000 // 0
204                      {0x00,  // 0b0000 // 0
205                      0x00,  // 0b0000 // 0
206                      0x00,  // 0b0000 // 0
207                      0x00,  // 0b0000 // 0
208                      0x00}, // 0b0000 // 0
209                      {0x00,  // 0b0000 // 0
210                      0x00,  // 0b0000 // 0
211                      0x00,  // 0b0000 // 0
212                      0x00,  // 0b0000 // 0
213                      0x00}, // 0b0000 // 0
214                      {0x00,  // 0b0000 // 0
215                      0x00,  // 0b0000 // 0
216                      0x00,  // 0b0000 // 0
217                      0x00,  // 0b0000 // 0
218                      0x00}, // 0b0000 // 0
219                      {0x00,  // 0b0000 // 0
220                      0x00,  // 0b0000 // 0
221                      0x00,  // 0b0000 // 0
222                      0x00,  // 0b0000 // 0
223                      0x00}, // 0b0000 // 0
224                      {0x00,  // 0b0000 // 0
225                      0x00,  // 0b0000 // 0
226                      0x00,  // 0b0000 // 0
227                      0x00,  // 0b0000 // 0
228                      0x00}, // 0b0000 // 0
229                      {0x00,  // 0b0000 // 0
230                      0x00,  // 0b0000 // 0
231                      0x00,  // 0b0000 // 0
232                      0x00,  // 0b0000 // 0
233                      0x00}, // 0b0000 // 0
234                      {0x00,  // 0b0000 // 0
235                      0x00,  // 0b0000 // 0
236                      0x00,  // 0b0000 // 0
237                      0x00,  // 0b0000 // 0
238                      0x00}, // 0b0000 // 0
239                      {0x00,  // 0b0000 // 0
240                      0x00,  // 0b0000 // 0
241                      0x00,  // 0b0000 // 0
242                      0x00,  // 0b0000 // 0
243                      0x00}, // 0b0000 // 0
244                      {0x00,  // 0b0000 // 0
245                      0x00,  // 0b0000 // 0
246                      0x00,  // 0b0000 // 0
247                      0x00,  // 0b0000 // 0
248                      0x00}, // 0b0000 // 0
249                      {0x00,  // 0b0000 // 0
250                      0x00,  // 0b0000 // 0
251                      0x00,  // 0b0000 // 0
252                      0x00,  // 0b0000 // 0
253                      0x00}, // 0b0000 // 0
254                      {0x00,  // 0b0000 // 0
255                      0x00,  // 0b0000 // 0
256                      0x00,  // 0b0000 // 0
257                      0x00,  // 0b0000 // 0
258                      0x00}, // 0b0000 // 0
259                      {0x00,  // 0b0000 // 0
260                      0x00,  // 0b0000 // 0
261                      0x00,  // 0b0000 // 0
262                      0x00,  // 0b0000 // 0
263                      0x00}, // 0b0000 // 0
264                      {0x00,  // 0b0000 // 0
265                      0x00,  // 0b0000 // 0
266                      0x00,  // 0b0000 // 0
267                      0x00,  // 0b0000 // 0
268                      0x00}, // 0b0000 // 0
269                      {0x00,  // 0b0000 // 0
270                      0x00,  // 0b0000 // 0
271                      0x00,  // 0b0000 // 0
272                      0x00,  // 0b0000 // 0
273                      0x00}, // 0b0000 // 0
274                      {0x00,  // 0b0000 // 0
275                      0x00,  // 0b0000 // 0
276                      0x00,  // 0b0000 // 0
277                      0x00,  // 0b0000 // 0
278                      0x00}, // 0b0000 // 0
279                      {0x00,  // 0b0000 // 0
280                      0x00,  // 0b0000 // 0
281                      0x00,  // 0b0000 // 0
282                      0x00,  // 0b0000 // 0
283                      0x00}, // 0b0000 // 0
284                      {0x00,  // 0b0000 // 0
285                      0x00,  // 0b0000 // 0
286                      0x00,  // 0b0000 // 0
287                      0x00,  // 0b0000 // 0
288                      0x00}, // 0b0000 // 0
289                      {0x00,  // 0b0000 // 0
290                      0x00,  // 0b0000 // 0
291                      0x00,  // 0b0000 // 0
292                      0x00,  // 0b0000 // 0
293                      0x00}, // 0b0000 // 0
294                      {0x00,  // 0b0000 // 0
295                      0x00,  // 0b0000 // 0
296                      0x00,  // 0b0000 // 0
297                      0x00,  // 0b0000 // 0
298                      0x00}, // 0b0000 // 0
299                      {0x00,  // 0b0000 // 0
300                      0x00,  // 0b0000 // 0
301                      0x00,  // 0b0000 // 0
302                      0x00,  // 0b0000 // 0
303                      0x00}, // 0b0000 // 0
304                      {0x00,  // 0b0000 // 0
305                      0x00,  // 0b0000 // 0
306                      0x00,  // 0b0000 // 0
307                      0x00,  // 0b0000 // 0
308                      0x00}, // 0b0000 // 0
309                      {0x00,  // 0b0000 // 0
310                      0x00,  // 0b0000 // 0
311                      0x00,  // 0b0000 // 0
312                      0x00,  // 0b0000 // 0
313                      0x00}, // 0b0000 // 0
314                      {0x00,  // 0b0000 // 0
315                      0x00,  // 0b0000 // 0
316                      0x00,  // 0b0000 // 0
317                      0x00,  // 0b0000 // 0
318                      0x00}, // 0b0000 // 0
319                      {0x00,  // 0b0000 // 0
320                      0x00,  // 0b0000 // 0
321                      0x00,  // 0b0000 // 0
322                      0x00,  // 0b0000 // 0
323                      0x00}, // 0b0000 // 0
324                      {0x00,  // 0b0000 // 0
325                      0x00,  // 0b0000 // 0
326                      0x00,  // 0b0000 // 0
327                      0x00,  // 0b0000 // 0
328                      0x00}, // 0b0000 // 0
329                      {0x00,  // 0b0000 // 0
330                      0x00,  // 0b0000 // 0
331                      0x00,  // 0b0000 // 0
332                      0x00,  // 0b0000 // 0
333                      0x00}, // 0b0000 // 0
334                      {0x00,  // 0b0000 // 0
335                      0x00,  // 0b0000 // 0
336                      0x00,  // 0b0000 // 0
337                      0x00,  // 0b0000 // 0
338                      0x00}, // 0b0000 // 0
339                      {0x00,  // 0b0000 // 0
340                      0x00,  // 0b0000 // 0
341                      0x00,  // 0b0000 // 0
342                      0x00,  // 0b0000 // 0
343                      0x00}, // 0b0000 // 0
344                      {0x00,  // 0b0000 // 0
345                      0x00,  // 0b0000 // 0
346                      0x00,  // 0b0000 // 0
347                      0x00,  // 0b0000 // 0
348                      0x00}, // 0b0000 // 0
349                      {0x00,  // 0b0000 // 0
350                      0x00,  // 0b0000 // 0
351                      0x00,  // 0b0000 // 0
352                      0x00,  // 0b0000 // 0
353                      0x00}, // 0b0000 // 0
354                      {0x00,  // 0b0000 // 0
355                      0x00,  // 0b0000 // 0
356                      0x00,  // 0b0000 // 0
357                      0x00,  // 0b0000 // 0
358                      0x00}, // 0b0000 // 0
359                      {0x00,  // 0b0000 // 0
360                      0x00,  // 0b0000 // 0
361                      0x00,  // 0b0000 // 0
362                      0x00,  // 0b0000 // 0
363                      0x00}, // 0b0000 // 0
364                      {0x00,  // 0b0000 // 0
365                      0x00,  // 0b0000 // 0
366                      0x00,  // 0b0000 // 0
367                      0x00,  // 0b0000 // 0
368                      0x00}, // 0b0000 // 0
369                      {0x00,  // 0b0000 // 0
370                      0x00,  // 0b0000 // 0
371                      0x00,  // 0b0000 // 0
372                      0x00,  // 0b0000 // 0
373                      0x00}, // 0b0000 // 0
374                      {0x00,  // 0b0000 // 0
375                      0x00,  // 0b0000 // 0
376                      0x00,  // 0b0000 // 0
377                      0x00,  // 0b0000 // 0
378                      0x00}, // 0b0000 // 0
379                      {0x00,  // 0b0000 // 0
380                      0x00,  // 0b0000 // 0
381                      0x00,  // 0b0000 // 0
382                      0x00,  // 0b0000 // 0
383                      0x00}, // 0b0000 // 0
384                      {0x00,  // 0b0000 // 0
385                      0x00,  // 0b0000 // 0
386                      0x00,  // 0b0000 // 0
387                      0x00,  // 0b0000 // 0
388                      0x00}, // 0b0000 // 0
389                      {0x00,  // 0b0000 // 0
390                      0x00,  // 0b0000 // 0
391                      0x00,  // 0b0000 // 0
392                      0x00,  // 0b0000 // 0
393                      0x00}, // 0b0000 // 0
394                      {0x00,  // 0b0000 // 0
395                      0x00,  // 0b0000 // 0
396                      0x00,  // 0b0000 // 0
397                      0x00,  // 0b0000 // 0
398                      0x00}, // 0b0000 // 0
399                      {0x00,  // 0b0000 // 0
400                      0x00,  // 0b0000 // 0
401                      0x00,  // 0b0000 // 0
402                      0x00,  // 0b0000 // 0
403                      0x00}, // 0b0000 // 0
404                      {0x00,  // 0b0000 // 0
405                      0x00,  // 0b0000 // 0
406                      0x00,  // 0b0000 // 0
407                      0x00,  // 0b0000 // 0
408                      0x00}, // 0b0000 // 0
409                      {0x00,  // 0b0000 // 0
410                      0x00,  // 0b0000 // 0
411                      0x00,  // 0b0000 // 0
412                      0x00,  // 0b0000 // 0
413                      0x00}, // 0b0000 // 0
414                      {0x00,  // 0b0000 // 0
415                      0x00,  // 0b0000 // 0
416                      0x00,  // 0b0000 // 0
417                      0x00,  // 0b0000 // 0
418                      0x00}, // 0b0000 // 0
419                      {0x00,  // 0b0000 // 0
420                      0x00,  // 0b0000 // 0
421                      0x00,  // 0b0000 // 0
422                      0x00,  // 0b0000 // 0
423                      0x00}, // 0b0000 // 0
424                      {0x00,  // 0b0000 // 0
425                      0x00,  // 0b0000 // 0
426                      0x00,  // 0b0000 // 0
427                      0x00,  // 0b0000 // 0
428                      0x00}, // 0b0000 // 0
429                      {0x00,  // 0b0000 // 0
430                      0x00,  // 0b0000 // 0
431                      0x00,  // 0b0000 // 0
432                      0x00,  // 0b0000 // 0
433                      0x00}, // 0b0000 // 0
434                      {0x00,  // 0b0000 // 0
435                      0x00,  // 0b0000 // 0
436                      0x00,  // 0b0000 // 0
437                      0x00,  // 0b0000 // 0
438                      0x00}, // 0b0000 // 0
439                      {0x00,  // 0b0000 // 0
440                      0x00,  // 0b0000 // 0
441                      0x00,  // 0b0000 // 0
442                      0x00,  // 0b0000 // 0
443                      0x00}, // 0b0000 // 0
444                      {0x00,  // 0b0000 // 0
445                      0x00,  // 0b0000 // 0
446                      0x00,  // 0b0000 // 0
447                      0x00,  // 0b0000 // 0
448                      0x00}, // 0b0000 // 0
449                      {0x00,  // 0b0000 // 0
450                      0x00,  // 0b0000 // 0
451                      0x00,  // 0b0000 // 0
452                      0x00,  // 0b0000 // 0
453                      0x00}, // 0b0000 // 0
454                      {0x00,  // 0b0000 // 0
455                      0x00,  // 0b0000 // 0
456                      0x00,  // 0b0000 // 0
457                      0x00,  // 0b0000 // 0
458                      0x00}, // 0b0000 // 0
459                      {0x00,  // 0b0000 // 0
460                      0x00,  // 0b0000 // 0
461                      0x00,  // 0b0000 // 0
462                      0x00,  // 0b0000 // 0
463                      0x00}, // 0b0000 // 0
464                      {0x00,  // 0b0000 // 0
465                      0x00,  // 0b0000 // 0
466                      0x00,  // 0b0000 // 0
467                      0x00,  // 0b0000 // 0
468                      0x00}, // 0b0000 // 0
469                      {0x00,  // 0b0000 // 0
470                      0x00,  // 0b0000 // 0
471                      0x00,  // 0b0000 // 0
472                      0x00,  // 0b0000 // 0
473                      0x00}, // 0b0000 // 0
474                      {0x00,  // 0b0000 // 0
475                      0x00,  // 0b0000 // 0
476                      0x00,  // 0b0000 // 0
477                      0x00,  // 0b0000 // 0
478                      0x00}, // 0b0000 // 0
479                      {0x00,  // 0b0000 // 0
480                      0x00,  // 0b0000 // 0
481                      0x00,  // 0b0000 // 0
482                      0x00,  // 0b0000 // 0
483                      0x00}, // 0b0000 // 0
484                      {0x00,  // 0b0000 // 0
485                      0x00,  // 0b0000 // 0
486                      0x00,  // 0b0000 // 0
487                      0x00,  // 0b0000 // 0
488                      0x00}, // 0b0000 // 0
489                      {0x00,  // 0b0000 // 0
490                      0x00,  // 0b0000 // 0
491                      0x00,  // 0b0000 // 0
492                      0x00,  // 0b0000 // 0
493                      0x00}, // 0b0000 // 0
494                      {0x00,  // 0b0000 // 0
495                      0x00,  // 0b0000 // 0
496                      0x00,  // 0b0000 // 0
497                      0x00,  // 0b0000 // 0
498                      0x00}, // 0b0000 // 0
499                      {0x00,  // 0b0000 // 0
500                      0x00,  // 0b0000 // 0
501                      0x00,  // 0b0000 // 0
502                      0x00,  // 0b0000 // 0
503                      0x00}, // 0b0000 // 0
504                      {0x00,  // 0b0000 // 0
505                      0x00,  // 0b0000 // 0
506                      0x00,  // 0b0000 // 0
507                      0x00,  // 0b0000 // 0
508                      0x00}, // 0b0000 // 0
509                      {0x00,  // 0b0000 // 0
510                      0x00,  // 0b0000 // 0
511                      0x00,  // 0b0000 // 0
512                      0x00,  // 0b0000 // 0
513                      0x00}, // 0b0000 // 0
514                      {0x00,  // 0b0000 // 0
515                      0x00,  // 0b0000 // 0
516                      0x00,  // 0b0000 // 0
517                      0x00,  // 0b0000 // 0
518                      0x00}, // 0b0000 // 0
519                      {0x00,  // 0b0000 // 0
520                      0x00,  // 0b0000 // 0
521                      0x00,  // 0b0000 // 0
522                      0x00,  // 0b0000 // 0
523                      0x00}, // 0b0000 // 0
524                      {0x00,  // 0b0000 // 0
525                      0x00,  // 0b0000 // 0
526                      0x00,  // 0b0000 // 0
527                      0x00,  // 0b0000 // 0
528                      0x00}, // 0b0000 // 0
529                      {0x00,  // 0b0000 // 0
530                      0x00,  // 0b0000 // 0
531                      0x00,  // 0b0000 // 0
532                      0x00,  // 0b0000 // 0
533                      0x00}, // 0b0000 // 0
534                      {0x00,  // 0b0000 // 0
535                      0x00,  // 0b0000 // 0
536                      0x00,  // 0b0000 // 0
537                      0x00,  // 0b0000 // 0
538                      0x00}, // 0b0000 // 0
539                      {0x00,  // 0b0000 // 0
540                      0x00,  // 0b0000 // 0
541                      0x00,  // 0b0000 // 0
542                      0x00,  // 0b0000 // 0
543                      0x00}, // 0b0000 // 0
544                      {0x00,  // 0b0000 // 0
545                      0x00,  // 0b0000 // 0
546                      0x00,  // 0b0000 // 0
547                      0x00,  // 0b0000 // 0
548                      0x00}, // 0b0000 // 0
549                      {0x00,  // 0b0000 // 0
550                      0x00,  // 0b0000 //
```

```
40          0x09, // 0b1001
41          0x09, // 0b1001
42          0x09, // 0b1001
43          0x0f}, // 0b1111
44
45          {0x01, // 0b0001 // 1
46          0x01, // 0b0001
47          0x01, // 0b0001
48          0x01, // 0b0001
49          0x01}, // 0b0001
50
51          {0x0f, // 0b1111 // 2
52          0x01, // 0b0001
53          0x0f, // 0b1111
54          0x08, // 0b1000
55          0x0f}, // 0b1111
56
57          {0x0f, // 0b1111 // 3
58          0x01, // 0b0001
59          0x0f, // 0b1111
60          0x01, // 0b0001
61          0x0f}, // 0b1111
62
63          {0x09, // 0b1001 // 4
64          0x09, // 0b1001
65          0x0f, // 0b1111
66          0x01, // 0b0001
67          0x01}, // 0b0001
68
69          {0x0f, // 0b1111 // 5
70          0x08, // 0b1000
71          0x0f, // 0b1111
72          0x01, // 0b0001
73          0x0f}, // 0b1111
74
75          {0x0f, // 0b1111 // 6
76          0x08, // 0b1000
77          0x0f, // 0b1111
78          0x09, // 0b1001
79          0x0f}, // 0b1111
80
81          {0x0f, // 0b1111 // 7
82          0x09, // 0b1001
83          0x09, // 0b1001
84          0x01, // 0b0001
85          0x01}, // 0b0001
86
87          {0x0f, // 0b1111 // 8
88          0x09, // 0b1001
89          0x0f, // 0b1111
90          0x09, // 0b1001
91          0x0f}, // 0b1111
92
```

```
93             {0x0f, // 0b1111 // 9
94             0x09, // 0b1001
95             0x0f, // 0b1111
96             0x01, // 0b0001
97             0x0f}));// 0b1111
98
99 // 定義した書体に従い、数を表示する
100 for(int n = 0; n <= 9; n++) {
101     // bit が立っているところを■、
102     // 立っていないところを□として、表示させます。
103     show_font(bitmap[n]);
104     printf("\n");
105 }
106
107 return 0;
108 }
```

3.5 作成例 ★★★★★

♣ ポーカー

▼ cards1.txt

```

1 ****
2 * カードの表現
3 * |A234567890JQK
4 * -----+
5 * Club |ABCDEFGHIJKLM
6 * Diamond|QRSTUVWXYZ[\]
7 * Heart |abcdefghijklm
8 * Spade |qrstuvwxyz{|}
9 *
10 * 例 ) Cards と書くと
11 * Cは Club(三つ葉)の3
12 * aは Heart(ハート)のA
13 * rは Spade(スペード)の2
14 * dは Heart(ハート)の4
15 * sは Spade(スペード)の3 を表すので
16 * ワンペアとなります。
17 ****
18 /* プレーヤー 1 が所有するカード */
19 Cards
20 /* プレーヤー 2 が所有するカード */
21 z{|}q

```

▼ poker.c

```

1 ****
2 //
3 // ポーカー
4 //
5 // https://ja.wikipedia.org/wiki/ポーカー
6 // https://ja.wikipedia.org/wiki/ポーカー・ハンドの一覧
7 //
8 // 文字コードとビット演算の学習を兼ねて、次のようにカードを表現します。
9 //
10 // ASCIIコード表(抜粋)
11 // |123456789ABCD
12 // -----
13 // 0x40 |ABCDEFGHIJKLM
14 // 0x50 |QRSTUVWXYZ[\]
15 // 0x60 |abcdefghijklm
16 // 0x70 |qrstuvwxyz{|}
17 //
18 // 文字'A'の文字コードは0x41です。
19 // 上位4bitでカードの種類(suits)を、
20 // 下位4bitでカードの数字(rank)を、表現することとします。
21 // するとトランプカード一式は次のように表現できます。
22 //
23 // カードの表現

```

```

24 // |A234567890JQK
25 // -----
26 // ♣Club |ABCDEFGHIJKLM
27 // ♦Diamond|QRSTU VWXYZ[\]
28 // ♥Heart |abcdefghijklm
29 // ♠Spade |qrstuvwxyz{|}
30 //
31 // 例) Cards と書くと
32 // Cは Club(三つ葉)の3
33 // aは Heart(ハート)のA
34 // rは Spade(スペード)の2
35 // dは Heart(ハート)の4
36 // sは Spade(スペード)の3 を表すので
37 // ワンペアとなります。
38 //
39 // プレイヤーの手札は文字型の配列で表現します。
40 // hands[1]:EJKLM
41 // hands[2]:ejklm
42 //
43 // また、役の判定を行いやすくするために、次のような管理配列を用意します。
44 // cards[6][17]
45 // | A 2 3 4 5 6 7 8 9 0 J Q K A | P1 P2
46 // -----
47 // Clubs | 0 0 0 0 1 0 0 0 0 1 1 1 1 0 | 5 0
48 // Diamonds | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 0
49 // Hearts | 0 0 0 0 2 0 0 0 0 2 2 2 2 0 | 0 5
50 // Spades | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 0
51 // -----
52 // Player 1 | 0 0 0 0 1 0 0 0 0 1 1 1 1 0 | 0 0
53 // Player 2 | 0 0 0 0 1 0 0 0 0 1 1 1 1 0 | 0 0
54 //
55 // この場合、プレーヤー1は三つ葉を5枚、プレーヤー2はハートを5枚持っています。
56 // ですので、どちらの役もフラッシュです。
57 //*****
58
59 #include <stdio.h> // 標準入出力関数
60 #include <stdlib.h> // exit関数
61 #include <string.h> // 文字列操作関数
62 #include <time.h> // 日付時刻の操作用関数
63
64 // 文字列用のバッファサイズ
65 #define BUFFER_SIZE 255
66 // デバッグを簡単にするためのマクロ定義
67 #define p(var) printf(#var ":%s\n", var)
68
69 // カードの種類
70 typedef enum {
71     CLUBS = 4,
72     DIAMONDS,
73     HEARTS,
74     SPADES
75 } suits;
76

```

```

77 // 役の種類
78 typedef enum {
79     NO_PAIR,           // 0 ぶた
80     ONE_PAIR,          // 1 ワンペア
81     TWO_PAIR,          // 2 ツーペア
82     THREE_OF_A_KIND, // 3 スリーカード
83     STRAIGHT,          // 4 ストレート
84     FLUSH,             // 5 フラッシュ
85     FULL_HOUSE,        // 6 フルハウス
86     FOUR_OF_A_KIND,   // 7 フォーカード
87     STRAIGHT_FLUSH,   // 8 ストレートフラッシュ
88     TOTAL_SCORE,       // 9 役判定後の合計得点
89 } hands_type;
90
91 // 役の名称
92 const char* HANDS_NAME[9] = {
93     "ぶた",
94     "ワンペア",
95     "ツーペア",
96     "スリーカード",
97     "ストレート",
98     "フラッシュ",
99     "フルハウス",
100    "フォーカード",
101    "ストレートフラッシュ",
102 };
103
104 // プロトタイプ宣言
105 // 初期化
106 void init_cards();
107 void init_hands();
108 void init_scores();
109
110 // 表示用
111 void disp_cards();
112 void disp_scores();
113 void draw_ascii_art(char mycard);
114
115 // 処理用
116 int set_card(char mycard, int player);
117 void calc_cards();
118 char get_card();
119 void calc_hands();
120 int my_random(int n);
121 void pause();
122 void usage(char const *argv[]);
123
124 // 役判定用
125 int is_no_pair(int player);
126 int is_one_pair(int player);
127 int is_two_pair(int player);
128 int is_three_of_a_kind(int player);
129 int is_straight(int player);

```

```

130 int is_flush(int player);
131 int is_four_of_a_kind(int player);
132 int get_suit(int player, int rank);
133
134 // グローバル変数
135 char hands[2+1][5+1];      // プレーヤーの手元にある5枚のカードを保持する配列
136                         // 添字の0は未使用
137 char cards[4+2][14+2+1]; // カード管理用配列
138                         // 10,J,Q,K,A のストレート判定の為
139                         // K の次にも Aを設けることで、14要素
140                         // Player1, 2の合計用に 2要素追加し
141                         // 添字0は未使用の為、17要素とする
142 int scores[2+1][9+1][2+1];// 役を点数に変換、強弱を判定するために用いる配列
143                         // 添字の0は未使用
144
145 /*-----*
146 // メイン関数
147 -----*/
148 int main(int argc, char const *argv[]) {
149     int n;                  // n枚目の手札
150     char str[BUFFER_SIZE + 1]; // ファイルから一行読み込むための作業用
151     FILE *fp;               // ファイルポインタ
152     int player;             // 1ならプレーヤー1、2ならプレーヤー2を示す
153     char mycard;            // 一枚場から引いたカード
154
155     // 初期化処理
156     init_cards();
157     init_hands();
158     init_scores();
159
160     // コマンドライン引数無なら、使い方表示して終了
161     if (argc < 2) {
162         usage(argv);
163         exit(0);
164     }
165
166     // -f ファイルからの読み取りオプションなら
167     if (strcmp(argv[1], "-f") == 0) {
168         if ((fp = fopen(argv[2], "r")) == NULL) {
169             // エラーメッセージ表示
170             printf("%s を開けませんでした。\\n", argv[2]);
171             exit(1);
172         }
173
174         player = 1;
175         while (1) {
176             // ファイルから手を読み込む
177             fgets(str, BUFFER_SIZE, fp);
178             // コメント行('*'や'/'を含む行)でなければ、
179             if (strchr(str, '*') == NULL && strchr(str, '/') == NULL) {
180                 // 読み取った手をセットする
181                 strcpy(hands[player], str);
182                 hands[player][5] = '\\0';

```

```

183     if (player == 1) {
184         player++;
185     } else {
186         break;
187     }
188 }
189 fclose(fp);
190
191 // 各プレーヤーが手にしたカード情報を、カード管理配列へ書き出す
192 for (player = 1; player <= 2; player++) {
193     // 0枚目から4枚目までのカードを順にセットする
194     for (n = 0; n < 5; n++) {
195         mycard = hands[player][n];
196         // ファイル入力時は、同じカードは所有していないはずなので、
197         // エラーチェックはしていない
198         set_card(mycard, player);
199     }
200 }
201
202 // -m 交互入力オプションなら
203 } else {
204     for (player = 1; player <= 2; player++) {
205         // 0枚目から4枚目までのカードを場から順に引く
206         for (n = 0; n < 5; n++) {
207             do {
208                 mycard = get_card();           // ランダムにカードを引く
209                 draw_ascii_art(mycard);    // 引いたカードをアスキーアートで表示
210                 pause();                  // 一時停止
211                 hands[player][n] = mycard; // 手札配列にセット
212             } while (set_card(mycard, player) != 0); // 既出のカードなら引き直す
213         }
214     }
215 }
216
217 // 各playerの手元にあるカードを表示
218 p(hands[1]);
219 p(hands[2]);
220
221 // 得点計算
222 calc_cards();
223
224 // 役を計算
225 calc_hands();
226
227 // カード管理配列の表示
228 disp_cards();
229
230 // 得点表示
231 disp_scores();
232
233 // 役を表示
234 for (player = 1; player <= 2; player++) {
235

```

```
236     int h = scores[player][TOTAL_SCORE][0] / 1000;
237     printf("プレーヤー%d の役は %s です\n", player, HANDS_NAME[h]);
238 }
239
240 // 勝敗表示
241 if (scores[1][TOTAL_SCORE][0] > scores[2][TOTAL_SCORE][0]) {
242     printf("プレーヤー1の勝ちです\n");
243 } else if (scores[1][TOTAL_SCORE][0] < scores[2][TOTAL_SCORE][0]) {
244     printf("プレーヤー2の勝ちです\n");
245 } else {
246     printf("引き分けです\n");
247 }
248 return 0;
249 }
250
251 /*-----
252 // カード配列の初期化
253 -----*/
254 void init_cards() {
255     int rank, suit;
256     for (suit = 0; suit < 6; suit++) {
257         for (rank = 0; rank < 17; rank++) {
258             cards[suit][rank] = 0;
259         }
260     }
261 }
262
263 /*-----
264 // 得点配列の初期化
265 -----*/
266 void init_scores() {
267     int hand, attr, player;
268     for (player = 0; player <= 2; player++) {
269         for (hand = NO_PAIR; hand <= TOTAL_SCORE; hand++) {
270             for (attr = 0; attr < 3; attr++) {
271                 scores[player][hand][attr] = 0;
272             }
273         }
274     }
275 }
276
277 /*-----
278 // プレーヤーの手にしているカードの初期化
279 -----*/
280 void init_hands() {
281     int player, n;
282     for (player = 0; player <= 2; player++) {
283         for (n = 0; n <= 5; n++) {
284             hands[player][n] = 0;
285         }
286     }
287 }
```

```

289 /*-----*/
290 // カードの表示
291 -----*/
292 void disp_cards() {
293     int suit, rank;
294     printf("\n");
295     printf("      | A 2 3 4 5 6 7 8 9 0 J Q K A | P1 P2\n");
296     printf("-----+-----+-----\n");
297     for (suit = 0; suit < 6; suit++) {
298         switch (suit) {
299             case 0:
300                 printf("Clubs    | "); break;
301             case 1:
302                 printf("Diamonds | "); break;
303             case 2:
304                 printf("Hearts   | "); break;
305             case 3:
306                 printf("Spades   | "); break;
307             case 4:
308                 printf("Player 1 | "); break;
309             case 5:
310                 printf("Player 2 | "); break;
311         }
312         for (rank = 1; rank < 17; rank++) {
313             printf("%ld ", cards[suit][rank]);
314             if (rank == 14) {
315                 printf("|\n");
316             }
317         }
318         printf("\n");
319         if (suit == 3) {
320             printf("-----+-----+-----\n");
321         }
322     }
323     printf("\n");
324 }
325
326 /*-----*/
327 // 得点の表示
328 -----*/
329 void disp_scores() {
330     int hand;
331     printf("          Player1          Player2      \n");
332     printf("          Score suit rank      Score suit rank\n");
333     printf("-----+-----+-----\n");
334     for (hand = TOTAL_SCORE; hand >= NO_PAIR; hand--) {
335         switch (hand) {
336             case TOTAL_SCORE:
337                 printf("= TOTAL =| "); break;
338             case STRAIGHT_FLUSH:
339                 printf("S.Flush | "); break;
340             case FOUR_OF_A_KIND:
341                 printf("4 cards | "); break;

```

```

342     case FULL_HOUSE:
343         printf("Full H. | "); break;
344     case FLUSH:
345         printf("Flush | "); break;
346     case STRAIGHT:
347         printf("Straight | "); break;
348     case THREE_OF_A_KIND:
349         printf("3 cards | "); break;
350     case TWO_PAIR:
351         printf("2 pair | "); break;
352     case ONE_PAIR:
353         printf("1 pair | "); break;
354     case NO_PAIR:
355         printf("no pair | "); break;
356     }
357     printf("%5d %5d %5d %5d %5d %5d\n",
358            scores[1][hand][0], scores[1][hand][2], scores[1][hand][1],
359            scores[2][hand][0], scores[2][hand][2], scores[2][hand][1]);
360   }
361   printf("\n");
362 }
363
364 /*-----
365 // カード管理配列へ、プレーヤーの持っているカードをセットする
366 -----*/
367 int set_card(char mycard, int player) {
368     int suit; // 三つ葉、ダイヤ、ハート、スペード
369     int rank; // トランプに書かれている数字
370
371     suit = ((mycard & 0xf0) >> 4) - 4;
372     rank = (mycard & 0x0f);
373
374     // 例 mycard = 'A' の場合
375     // suits = ((mycard & 0xf0) >> 4) - 4;
376     // 'A' = 0x41 = 0b0100_0001;
377     // 'A' & 0xf0
378     // 0x41 & 0xf0
379     // 0b0100_0001
380     // &) 0b1111_0000
381     // -----
382     // 0b0100_0000
383
384     // 0b0100_0000 >> 4 // 右へ4ビットシフト
385     // 0b0000_0100
386     // 0b0000_0100 - 4 = 0; // suits = 0となる
387
388     // mycard & 0x0f;
389     // 'A' = 0x41 = 0b0100_0001;
390     // 'A' & 0x0f
391     // 0x41 & 0x0f
392     // 0b0100_0001
393     // &) 0b0000_1111
394     // -----

```

```

395 // 0b0000_0001 // 下4ビットを取り出すことが出来た
396 // 0b0000_0001 = 1 ので rank(トランプの数字)は1と判明する
397
398 // まだどのプレーヤーの手札にもなっていなければ
399 if (cards[suit][rank] == 0) {
400     // playerの手札になっている旨、セットする
401     cards[suit][rank] = player;
402     if (rank == 1) {
403         cards[suit][14] = player; //エースの時
404     }
405     return 0; //正常終了
406 } else {
407     // 既にいずれかのプレーヤーの手札になっているならば
408     return cards[suit][rank]; //どのプレーヤーで用いられているか返す
409 }
410 }
411
412 /*
413 // cards配列内の縦横集計を行う
414 //          | A 2 3 4 5 6 7 8 9 0 J Q K A | P1 P2
415 // -----+-----+
416 // Clubs    | 0 0 0 0 0 0 0 0 0 1 1 1 1 0 | 5 0
417 // Diamonds | 0 0 0 0 0 0 0 0 0 0 0 1 0 | 0 0
418 // Hearts   | 2 0 0 0 0 0 0 0 0 2 2 2 2 2 | 0 5 <- Player2が5枚の
419 // Spades   | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 0      ♡所持を示す
420 // -----+-----+
421 // Player 1 | 0 0 0 0 0 0 0 0 0 1 1 1 2 0 | 0 0
422 // Player 2 | 1 0 0 0 0 0 0 0 0 1 1 1 1 1 | 0 14 <- Player2は 14(=A)から
423                                ^ 始まるフラッシュを意味する
424                                |__ Player1は2枚のK所持を示す
425 -----
426 void calc_cards() {
427     int rank;
428     int suit;
429     int sum;
430     int player;
431     int range;
432
433     // フラッシュか判定できるよう、右端集計欄に書き込む
434     for (player = 1; player <= 2; player++) {
435         for (suit = 0; suit <= 3; suit++) {
436             sum = 0;
437             for (rank = 1; rank <= 13; rank++) {
438                 if (cards[suit][rank] == player) {
439                     sum += (cards[suit][rank] / player);
440                 }
441             }
442             cards[suit][14 + player] = sum;
443         }
444     }
445
446     // フラッシュだった場合、右下隅に最高rankを書き込む
447     for (player = 1; player <= 2; player++) {

```

```

448 for (suit = 0; suit <= 3; suit++) {
449     if (cards[suit][14 + player] == 5) {
450         for (rank = 14; rank >= 5; rank--) {
451             if (cards[suit][rank] == player) {
452                 cards[suit][14 + player] = rank;
453                 break;
454             }
455         }
456     }
457 }
458 }
459
// 1ペア～4ペアまで判定しやすいよう、下の合計欄に書き込む
460 for (player = 1; player <= 2; player++) {
461     for (rank = 1; rank <= 14; rank++) {
462         sum = 0;
463         for (suit = 0; suit <= 3; suit++) {
464             if (cards[suit][rank] == player) {
465                 sum += (cards[suit][rank] / player);
466             }
467         }
468         cards[3 + player][rank] = sum;
469     }
470 }
471 }
472
// ストレートだった場合、右下隅に最高rankを書き込む
473 for (player = 1; player <= 2; player++) {
474     for (rank = 10; rank >= 1; rank--) {
475         sum = 0;
476         // 11111と連続する5つの範囲の合計を取る
477         for (range = 4; range >= 0; range--) {
478             if (cards[3 + player][rank + range] == 1) {
479                 sum += cards[3 + player][rank + range];
480             }
481         }
482         if (sum == 5) {
483             cards[3 + player][14 + player] = rank + 4;
484             break;
485         }
486     }
487 }
488 }
489 }
490
/*-----
491 // 役が成立しているか確認し、scores配列へ結果を書き込む
492 -----*/
493 void calc_hands() {
494     int rank;
495     int suit;
496     int player;
497     int hand;
498
499     for (player = 1; player <= 2; player++) {

```

```

501 rank = is_four_of_a_kind(player);
502 suit = get_suit(player, rank);
503 scores[player][FOUR_OF_A_KIND][1] = rank;
504 scores[player][FOUR_OF_A_KIND][2] = suit;
505
506 rank = is_flush(player);
507 suit = rank % 10;
508 scores[player][FLUSH][1] = rank / 10;
509 scores[player][FLUSH][2] = suit;
510
511 rank = is_straight(player);
512 suit = get_suit(player, rank);
513 scores[player][STRAIGHT][1] = rank;
514 scores[player][STRAIGHT][2] = suit;
515
516 rank = is_three_of_a_kind(player);
517 suit = get_suit(player, rank);
518 scores[player][THREE_OF_A_KIND][1] = rank;
519 scores[player][THREE_OF_A_KIND][2] = suit;
520
521 rank = is_two_pair(player);
522 suit = get_suit(player, rank);
523 scores[player][TWO_PAIR][1] = rank;
524 scores[player][TWO_PAIR][2] = suit;
525
526 rank = is_one_pair(player);
527 suit = get_suit(player, rank);
528 scores[player][ONE_PAIR][1] = rank;
529 scores[player][ONE_PAIR][2] = suit;
530
531 rank = is_no_pair(player);
532 suit = get_suit(player, rank);
533 scores[player][NO_PAIR][1] = rank;
534 scores[player][NO_PAIR][2] = suit;
535
536 // フルハウス ?
537 if (scores[player][THREE_OF_A_KIND][1] != 0 && scores[player][ONE_PAIR][1] != 0)
538 > {
539   scores[player][FULL_HOUSE][1] = scores[player][THREE_OF_A_KIND][1];
540   scores[player][FULL_HOUSE][2] = scores[player][THREE_OF_A_KIND][2];
541 }
542
543 // ストレートフラッシュ ?
544 if (scores[player][STRAIGHT][1] != 0 && scores[player][FLUSH][1] != 0) {
545   scores[player][STRAIGHT_FLUSH][1] = scores[player][FLUSH][1];
546   scores[player][STRAIGHT_FLUSH][2] = scores[player][FLUSH][2];
547 }
548
549 for (hand = STRAIGHT_FLUSH; hand >= NO_PAIR; hand--) {
550   // 十の位はrank、一の位はsuitとすることで、
551   // 同じランクであっても、大小関係から強弱を判定できる
552   scores[player][hand][0] = scores[player][hand][1] * 10 + scores[player][hand]>
553   [2];

```

```

552 }
553
554     for (hand = STRAIGHT_FLUSH; hand >= NO_PAIR; hand--) {
555         if (scores[player][hand][0] != 0) {
556             scores[player][TOTAL_SCORE][0] = scores[player][hand][0] + hand * 1000;
557             scores[player][TOTAL_SCORE][1] = scores[player][hand][1];
558             scores[player][TOTAL_SCORE][2] = scores[player][hand][2];
559             break;
560         }
561     }
562 }
563 }
564
565 /*-----
566 // フラッシュか判定
567 -----*/
568 int is_flush(int player) {
569     int suit;
570     for (suit = 0; suit <= 3; suit++) {
571         if (cards[suit][14 + player] >= 5) {
572             return cards[suit][14 + player] * 10 + suit + 1;
573         }
574     }
575     return 0; // フラッシュではなかった
576 }
577
578 /*-----
579 // ストレートか判定 rankを返す
580 -----*/
581 int is_straight(int player) {
582     // calc_cards で計算済みなので、値を返すのみ
583     return cards[3 + player][14 + player];
584 }
585
586 /*-----
587 // フォーカードか判定 rankを返す
588 -----*/
589 int is_four_of_a_kind(int player) {
590     int rank;
591     for (rank = 14; rank >= 2; rank--) {
592         if (cards[3 + player][rank] == 4) {
593             return rank;
594         }
595     }
596     return 0;
597 }
598
599 /*-----
600 // スリーカードか判定 rankを返す
601 -----*/
602 int is_three_of_a_kind(int player) {
603     int rank;
604     for (rank = 14; rank >= 2; rank--) {

```

```
605     if (cards[3 + player][rank] == 3) {
606         return rank;
607     }
608 }
609 return 0;
610 }

612 -----
613 // ツーペアか判定 rankを返す
614 -----
615 int is_two_pair(int player) {
616     int rank;
617     int max_rank = 0; // 発見したツーペアの内、ランクの高いもの
618     int count = 0;    // 発見したワンペアの数
619     for (rank = 14; rank >= 2; rank--) {
620         if (cards[3 + player][rank] == 2) {
621             if (max_rank < rank) {
622                 max_rank = rank;
623             }
624             count++;
625         }
626     }
627
628 // ツーペアならば
629 if (count == 2) {
630     return max_rank;
631 }
632 return 0;
633 }

635 -----
636 // ワンペアか判定 rankを返す
637 -----
638 int is_one_pair(int player) {
639     int rank;
640     for (rank = 14; rank >= 2; rank--) {
641         if (cards[3 + player][rank] == 2) {
642             return rank;
643         }
644     }
645     return 0;
646 }

648 -----
649 // ノーペアか判定
650 -----
651 int is_no_pair(int player) {
652     int rank;
653     for (rank = 14; rank >= 2; rank--) {
654         if (cards[3 + player][rank] == 1) {
655             return rank;
656         }
657     }
```

```
658     return 0; // error
659 }
660
661 /*-----
662 // 何のマークか？
663 // rank == 13の時、13は何のマークかを返す
664 -----*/
665 int get_suit(int player, int rank) {
666     int i;
667     for (i = 3; i >= 0; i--) {
668         if (cards[i][rank] == player) {
669             return i + 1;
670         }
671     }
672     return 0; // rankの数のカードは持っていない
673 }
674
675 /*-----
676 // アスキーアートでカードを表示
677 -----*/
678 void draw_ascii_art(char mycard) {
679     int suit; // 三つ葉、ダイヤ、ハート、スペード
680     char rank; // トランプに書かれている数字
681
682     suit = (mycard & 0xf0) >> 4;
683     rank = (mycard & 0x0f);
684
685     switch (rank) {
686     case 11: rank = 'J'; break;
687     case 12: rank = 'Q'; break;
688     case 13: rank = 'K'; break;
689     case 14: rank = 'A'; break;
690     case 1: rank = 'A'; break;
691     }
692
693     switch (suit) {
694     case CLUBS:
695         if (2 <= rank && rank <= 10) {
696             printf("      ▩      \n");
697             printf("      ( )      \n");
698             printf("      / %2d \\" , rank);
699             printf("C_____▷\n");
700             printf("      ^      \n");
701             printf("      / \\"      \n");
702         } else {
703             printf("      ▩      \n");
704             printf("      ( )      \n");
705             printf("      / %c \\" , rank);
706             printf("C_____▷\n");
707             printf("      ^      \n");
708             printf("      / \\"      \n");
709         }
710     break;
```

```

711 case DIAMONDS:
712     if (2 <= rank && rank <= 10) {
713         printf(" \n");
714         printf(" \n");
715         printf(" %2d \n", rank);
716         printf(" \n");
717         printf(" \n");
718         printf(" \n");
719     } else {
720         printf(" \n");
721         printf(" \n");
722         printf(" %c \n", rank);
723         printf(" \n");
724         printf(" \n");
725         printf(" \n");
726     }
727     break;
728 case HEARTS:
729     if (2 <= rank && rank <= 10) {
730         printf(" \n");
731         printf(" | \n");
732         printf(" | %2d | \n", rank);
733         printf(" | \n");
734         printf(" | \n");
735         printf(" | \n");
736     } else {
737         printf(" \n");
738         printf(" | \n");
739         printf(" | %c | \n", rank);
740         printf(" | \n");
741         printf(" | \n");
742         printf(" | \n");
743     }
744     break;
745 case SPADES:
746     if (2 <= rank && rank <= 10) {
747         printf(" \n");
748         printf(" \n");
749         printf(" %2d \n", rank);
750         printf(" | \n");
751         printf(" | \n");
752         printf(" | \n");
753     } else {
754         printf(" \n");
755         printf(" \n");
756         printf(" %c \n", rank);
757         printf(" | \n");
758         printf(" | \n");
759         printf(" | \n");
760     }
761     break;
762 }
763 }
```

```

764 /*
765 // 0-n未満の数を返す
766 */
767 int my_random(int n) {
768     return clock() % n;
769 }
770 /*
771 // カードをランダムに引く
772 */
773 char get_card() {
774     int suit; // 三つ葉、ダイヤ、ハート、スペード
775     int rank; // トランプに書かれている数字
776
777     suit = my_random(4);
778     rank = my_random(13) + 1;
779     return ((suit + 4) << 4) + rank;
780 }
781 /*
782 // 一時停止
783 */
784 void pause() {
785     printf("\n === press return key === \n");
786     while (getchar() != '\n')
787         ;
788 }
789 /*
790 // 使い方表示
791 */
792 void usage(char const *argv[]) {
793     printf(" --- 使い方 --- \n");
794     printf("1) %s -f cards.txt\n", argv[0]);
795     printf("指定されたファイル cards.txt からプレーヤーの手を読み込みます\n");
796     printf("\n");
797     printf("2) %s -m\n", argv[0]);
798     printf("交互にランダムでカードを引きます\n");
799     printf("\n");
800 }
801 */
802 }
```

♣ 双六ゲーム

▼ sugoroku.c

```

1 ****
2 * 双六ゲームを創ってみましょう。
3 * 止まった升目に「3つ進む」や、「振り出しに戻る」も創ってみましょう。
4 * どこまで進んだか分かる表示機能や、
5 * オープニング・エンディングもあると楽しいですね。*
6 ****
7 
```

```

8 #include "mt.h"      // Mersenne Twister法による乱数
9 #include <stdio.h>   // 標準入出力関数
10 #include <string.h>  // 文字列操作関数
11 #include <unistd.h>  // sleep関数
12
13 #define MAP_SIZE     22 // 0-21までの升目がある
14 #define GOAL_POSITION 21 // 21の升目が、上がり
15 #define TRUE          1 // 真
16 #define FALSE         0 // 偽
17
18 // 双六のマップ配置
19 // 0: スタート
20 // 1:
21 // 2:
22 // 3:
23 // 4: 2マス進む
24 // 5: 3マス戻る
25 // 6:
26 // 7: スタートに戻る
27 // 8: 2マス進む
28 // 9: 1回休み
29 // 10: 3マス戻る
30 // 11:
31 // 12: 2マス進む
32 // 13:
33 // 14: スタートに戻る
34 // 15: 3マス戻る
35 // 16: 2マス進む
36 // 17:
37 // 18: 2回休み
38 // 19:
39 // 20: 3マス戻る
40 // 21: ゴール
41
42 // main関数の外側で宣言することにより、
43 // 大域変数（グローバル変数）になります。
44 // 各関数から共通して見えるようになるので、プログラミングが楽になります。
45 // どこからでも参照・変更できる利便性の反面、
46 // 更新履歴を追い難く、デバッグが困難になる不利益もあるので、
47 // 一般的には好ましくないとされています。
48 // ここでは、双六作成が容易となるため、
49 // グローバル変数を許容することとします。
50 char const *map_event[] = {"", "", "", "", "2A", "3B", "", "SB",
51                 "", "1R", "3B", "", "2A", "", "SB", "3B",
52                 "2A", "", "2R", "", "3B", ""};
53
54 // #define PLAYER 0
55 // #define COMPUTER 1 と定義しても同様です。
56 // 列挙型(enum) や 型定義	typedef をご紹介したかったので、使ってています。
57
58 // 列挙型 cmp(競技者)型の宣言
59 typedef enum { PLAYER, COMPUTER } cmp;
60

```

```

61 char const *NAMES[2] = {"PLAYER", "COMPUTER"}; // 競技者の名称
62 char const initials[2] = {'P', 'C'}; // それぞれのイニシャル
63
64 // PLAYER, COMPUTER それぞれが、双六上のどの升目にいるか？
65 int position[2] = {};
66 // PLAYER, COMPUTER 何回休みか？
67 char rest[2];
68
69 // 関数のプロトタイプ宣言に代えて、直接、関数本体を記述します。
70 // その後、main 関数を記述します。
71
72
73 /* 双六描画
74 -----
75 void draw_map() {
76     int i;
77     cmp competitor; // cmp型変数 competitorの宣言
78
79     puts("");
80     puts(" ST 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 GL");
81     puts("+-+-+-+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+");
82
83     for (competitor = PLAYER; competitor <= COMPUTER; competitor++) {
84         for (i = 0; i < MAP_SIZE; i++) {
85             printf("| ");
86             if (position[competitor] == i) {
87                 printf("%c", initials[competitor]);
88             } else {
89                 printf(" ");
90             }
91         }
92         printf("|\n");
93     }
94
95     // event
96     puts("+-+-+-+---+---+---+---+---+---+---+---+---+---+---+---+---+---+");
97     puts(" | | | |2A|3B| |SB| |1R|3B| |2A| |SB|3B|2A| |2R| |3B| |");
98     puts("+-+-+-+---+---+---+---+---+---+---+---+---+---+---+---+---+---+");
99     puts("");
100    puts("【凡例】A: 進む B: 戻る R: 休み S:振り出し");
101    puts("");
102 }
103
104 /* 開始画面描画
105 -----
106 void draw_opening() {
107     // sleep(1)は一秒間、時の経過を待ちます。
108     puts("The Japanese Traditional Game "); sleep(1);
109     puts(" "); sleep(1);
110     puts(" ##### # # ##### ##### ##### # # # # "); sleep(1);
111     puts(" # # # # # # # # # # # # # # "); sleep(1);
112     puts(" # # # # # # # # # # # # # # "); sleep(1);
113     puts(" ##### # # # # ##### # # # # "); sleep(1);

```

```

114     puts("      # #      # #      # #      # #      # #      # #      #      #"); sleep(1);
115     puts("      # ##    ## #    # #    # #    # #    # #    # ##    ##"); sleep(1);
116     puts("#####  #####  ###  ##### #    #  ##### #    # ##### "); sleep(1);
117 }
118
119 /* 終了画面描画
120 -----
121 void draw_ending() {
122     puts("The Japanese Traditional Game          "); sleep(1);
123     puts("                                "); sleep(1);
124     puts("          S   U   G   O   R   O   K   U          "); sleep(1);
125     puts("                                "); sleep(1);
126     puts("          Fin.          "); sleep(1);
127 }
128
129 /* イベント処理
130 * 双六上の各イベントを受け取り、競技者の内部状態を適宜変更する
131 * | | | | |2A|3B| |SB| |1R|3B| |2A| |SB|3B|2A| |2R| |3B|
132 * 凡例: nV.
133 * n: 回数
134 * V: 動作. go Ahead(進む) / go Backward(戻る) / Rest(休み)
135 * 2A なら 二歩進む
136 * 3B なら 三歩戻る
137 * SB なら スタートまで戻る
138 * 1R なら 一回休み
139 -----
140 void event(cmp competitor) {
141     // イベントを取得
142     char event[3];
143     strcpy(event, map_event[position[competitor]]);
144     if (strlen(event) == 0) {
145         // 何もイベントはなかったとして終了
146         return;
147     }
148
149     // 該当イベントの処理
150     char times = event[0]; // 何歩進む、何回休みなどの数
151     char verbs = event[1]; // 進む、戻る、休むの種類
152     switch (verbs) {
153         // 進む
154         case 'A':
155             position[competitor] += (times - '0'); // 文字を数値に変換
156             // ゴールを通り過ぎていたら、ゴール地点に調整
157             if (position[competitor] > GOAL_POSITION) {
158                 position[competitor] = GOAL_POSITION;
159             }
160             // メッセージ表示
161             printf("%c> やった～ %c マス 進んだ！\n", initials[competitor], times);
162             break;
163
164         // 戻る
165         case 'B':
166             if (times == 'S') {

```

```
167     // 振り出しに戻る
168     position[competitor] = 0;
169 } else {
170     // 戻る
171     position[competitor] -= (times - '0'); // 文字を数値に変換
172     // スタートを通り過ぎていたら、スタート地点に調整
173     if (position[competitor] < 0) {
174         position[competitor] = 0;
175     }
176 }
177
178 // メッセージ表示
179 if (times == 'S') {
180     printf("%c> わ~ん スタートに 戻ったよ\n", initials[competitor]);
181 } else {
182     printf("%c> わ~ん %c マス 戻ったよ\n", initials[competitor], times);
183 }
184 break;
185
186 // 休み
187 case 'R':
188     rest[competitor] = (times - '0'); // 文字を数値に変換
189
190     // メッセージ表示
191     printf("%c> わ~ん %c 回 休みだよ\n", initials[competitor], times);
192     break;
193 }
194 }
195
196 /* さいころを振って進む
-----
197 void dice_and_walk(cmp competitor) {
198     // さいころを振って進む
199     int dice = genrand_int32() % 6 + 1;
200     position[competitor] += dice;
201
202     // ゴールを通り過ぎていたら、ゴール地点に調整
203     if (position[competitor] > GOAL_POSITION) {
204         position[competitor] = GOAL_POSITION;
205     }
206
207     // メッセージ表示
208     printf("%c> やった～ %d マス 進んだ！\n", initials[competitor], dice);
209 }
210
211
212 /* 双六を上がったか、判定関数
-----
213 int is_goal(cmp competitor) {
214     if (position[competitor] == GOAL_POSITION) {
215         return TRUE;
216     } else {
217         return FALSE;
218     }
219 }
```

```

220 }
221
222 /* メッセージを表示し、キー入力されるまで待機
223 -----
224 void message_and_wait(char const *message){
225     // メッセージの表示
226     printf("%s", message);
227     // キー入力されるまで待機
228     while (getchar() != '\n')
229     ;
230 }
231
232 /* メイン関数
233 -----
234 int main(int argc, char const *argv[]) {
235     cmp competitor = PLAYER; // cmp型変数 competitorの宣言と初期化
236     char message[30];         // メッセージ作成用配列を宣言
237
238     // タイトル表示
239     draw_opening();
240
241     // 双六表示
242     draw_map();
243
244     // 何かキーを押すと開始
245     message_and_wait("\nPress Enter to Start\n");
246
247     // 競技開始
248     do {
249         // message = "\nPLAYERの番 Press Enter\n";
250         // message = "\nCOMPUTERの番 Press Enter\n";
251         // と書けないので、文字配列を初期化し、表示させたい文字列を連結している。
252         memset(message, '\0', sizeof(message)); // 初期化
253         strcat(message, "\n");
254         strcat(message, NAMES[competitor]);
255         strcat(message, "の番 Press Enter\n");
256         message_and_wait(message);
257
258         // ○回休みでなければ
259         if (rest[competitor] == 0) {
260             // さいころを振って進む
261             dice_and_walk(competitor);
262             // 止まった升目に、イベントが設定されているか
263             event(competitor);
264             // お休み回数を減らす
265         } else {
266             printf("%c> %d 回休みなので進めない・・・\n",
267                   initials[competitor], rest[competitor]);
268             rest[competitor]--;
269         }
270
271         // 双六表示
272         draw_map();

```

```

273 // 次の競技者の番にする
274 competitor = (competitor + 1) % 2;
275
276 // どちらかが上がるまで、続行
277 } while (!is_goal(PLAYER) && !is_goal(COMPUTER));
278
279 // ゴール到着時のメッセージ
280 if (is_goal(competitor)) {
281     printf("%c> 双六を上がったよ(*^_^*)\n", initials[competitor]);
282 }
283
284 // エンディングのご案内
285 message_and_wait("\nPress Enter to Ending\n");
286
287 // エンディング表示
288 draw_ending();
289
290
291 return 0;
292 }
```

♣ マージソート（併合並び替え）

▼ linked_list.h

```

1  /*
2   * =====
3   * 単方向連結リストの実装のために、
4   * 定数、構造体、及び、挿入/削除/表示/解放関数の定義
5   * =====
6
7 #define NOT_FOUND -1
8 #define SUCCESS    1
9 #define FAILURE    0
10
11 // リスト用構造体定義
12 typedef struct list {
13     struct list *next;
14     int value;
15 } list_t;
16
17 // ノードの挿入
18 int insert_node(list_t *list, int number) {
19     // 新規確保用
20     list_t *p;
21     // 次のノードへのポインタ
22     list_t *next;
23     // 先頭から順にたどってきた最後のノード
24     list_t *last;
25
26     // 新しいリストの領域を確保
27     p = (list_t *)malloc(sizeof(list_t));
28
29     // メモリ確保に失敗
```

```
29 if (p == NULL) {
30     return FAILURE;
31 }
32
33 // 値を代入
34 p->value = number;
35 // 次の要素は末尾と分かるようにNULLを入れる。(番兵)
36 p->next = NULL;
37
38 // 先頭が末尾直前のポインタになる
39 last = list;
40
41 // 先頭ノードから順に末尾のノードまで移動
42 for (next = (*list).next; next != NULL; next = next->next) {
43     last = next;
44 }
45
46 // リストを連結する。
47 last->next = p;
48
49 return SUCCESS;
50 }
51
52 // ノードの削除
53 int remove_node(list_t *list, int number) {
54     list_t *p;
55
56     // 削除要素の直前のノードへのポインタ
57     list_t *prev;
58
59     // 最初は先頭要素の次のリストからチェックしてるので、
60     // 削除要素の直前の要素は先頭要素になる。
61     prev = list;
62
63     // リストを末尾(NULLになる)までループ
64     for (p = (*list).next; p != NULL; p = p->next) {
65         // その値があれば
66         if (p->value == number) {
67             // 削除要素の前のリストにつなげる
68             // 次の要素が末尾(NULL)なら、つなげる必要がないので事前にチェック
69             if (p->next != NULL) {
70                 // 削除直前の要素につなげる
71                 prev->next = p->next;
72                 // 削除対象要素の解放
73                 free(p);
74                 return SUCCESS;
75             }
76             // 削除要素が末尾の要素だった場合の処理
77             // 末尾要素にNULLを入れる
78             prev->next = NULL;
79
80             // 削除対象要素の解放
81             free(p);

```

```

82     return SUCCESS;
83 }
84     prev = p;
85 }
86     return NOT_FOUND;
87 }
88
89 // リストの表示
90 int show_list(list_t *list) {
91     if ((*list).next == NULL) {
92         return NOT_FOUND;
93     }
94
95     // NULLになるまで全部表示
96     list_t *p;
97     for (p = (*list).next; p != NULL; p = p->next) {
98         printf("%d ", p->value);
99     }
100    printf("\n");
101
102    return SUCCESS;
103 }
104
105 // リストの解放
106 void release_list(list_t *list) {
107     // 次のリストのポインタ
108     list_t *next;
109     // 削除対象のポインタ
110     list_t *delete_node;
111     next = (*list).next;
112
113     // NULLになるまでループ
114     while (next) {
115         // 削除対象のポインタを保存
116         delete_node = next;
117         // 次のリストのポインタを取得
118         next = next->next;
119         free(delete_node);
120     }
121 }
```

▼ linked_list.c

```

1 ****
2 * 連結リスト
3 * https://ja.wikipedia.org/wiki/連結リスト
4 *
5 * マージソート
6 * https://ja.wikipedia.org/wiki/マージソート
7 * https://programming-place.net/ppp/contents/algoritm/sort/007.html
8 ****
9 #include <stdio.h>
10 #include <stdlib.h>
```

```
11 #include "linked_list.h"
12
13 int main(int argc, char const *argv[]) {
14     // 線形リスト宣言と、一番最初のノード生成
15     list_t *list = (list_t *)malloc(sizeof(list_t));
16
17     // このリストにあるノードは一つだけで、
18     // 次の要素はまだ空なのでNULLを入れる（番兵）
19     list->next = NULL;
20     list->value = 0;
21
22     char answer;
23     int number;
24
25     while (1) {
26         printf("\n何をしますか？ 0.終了、1.追加、2.削除、3.表示\n");
27         scanf("%c", &answer);
28         while (getchar() != '\n')
29             ;
30
31         switch (answer) {
32             case '0':
33                 // リストの解放
34                 release_list(list);
35                 return 0;
36             case '1':
37                 printf("追加する値を入力して下さい> ");
38                 scanf("%d", &number);
39                 while (getchar() != '\n')
40                     ;
41                 if (insert_node(list, number) == SUCCESS) {
42                     printf("追加しました\n");
43                 } else {
44                     printf("追加用メモリが確保出来ませんでした\n");
45                 }
46                 break;
47             case '2':
48                 printf("削除する値を入力して下さい> ");
49                 scanf("%d", &number);
50                 while (getchar() != '\n')
51                     ;
52                 if (remove_node(list, number) == SUCCESS) {
53                     printf("削除しました\n");
54                 } else {
55                     printf("その値を持つノードは見つかりませんでした\n");
56                 }
57                 break;
58             case '3':
59                 if (show_list(list) == NOT_FOUND) {
60                     printf("まだ何もありません\n");
61                 }
62                 break;
63             default:
```

```

64     printf("正しい選択肢を入力して下さい\n");
65     break;
66   }
67 }
68 }
```

▼merge_sort.c

```

1 ****
2 * 連結リスト
3 * https://ja.wikipedia.org/wiki/連結リスト
4 *
5 * マージソート
6 * https://ja.wikipedia.org/wiki/マージソート
7 * https://programming-place.net/ppp/contents/algorithm/sort/007.html
8 ****
9 #include <stdio.h>
10 #include <stdlib.h>
11 #include "linked_list.h"
12
13 // 関数プロトタイプ宣言
14 list_t *merge(list_t *a, list_t *b);
15 list_t *merge_sort(list_t *list);
16 // void merge_sort(list_t *list);
17
18 int main(int argc, char const *argv[]) {
19   // 線形リスト宣言と、一番最初のノード生成
20   list_t *list = (list_t *)malloc(sizeof(list_t));
21
22   // このリストにあるノードは一つだけで、
23   // 次の要素はまだ空なのでNULLを入れる(番兵)
24   list->next = NULL;
25   list->value = 0;
26
27   // 初期値投入
28   insert_node(list, 3);
29   insert_node(list, 14);
30   insert_node(list, 15);
31   insert_node(list, 92);
32   insert_node(list, 65);
33   insert_node(list, 35);
34   insert_node(list, 89);
35   insert_node(list, 79);
36   insert_node(list, 24);
37   insert_node(list, 58);
38
39   // リストの表示
40   printf("並び替え前\n");
41   show_list(list);
42
43   // マージソート実行
44   merge_sort(list);
45 }
```

```

46 // リストの表示
47 printf("並び替え後\n");
48 show_list(list);
49
50 // リストの解放
51 release_list(list);
52
53 return 0;
54 }
55
56 // 二つのリストを併合(マージ)する
57 list_t *merge(list_t *a, list_t *b) {
58     list_t result;           // 併合されたリスト
59     list_t *w = &result;    // work 変数
60                         // これを起点に、a, b 小さい方を順に連結していく
61
62 // リストをマージ
63 while (a != NULL && b != NULL) {
64     // 昇順ソートのため、小さい要素を先にする。
65     // a と b が等しい場合は、先頭を優先すると安定ソートとなる。
66     if (a->value <= b->value) {
67         w->next = a;          // wの後ろにaを連結する
68         w      = w->next;    // aを連結したので、さらに小さい数を連結できるよう
69                         // 次のノードに進む
70         a      = a->next;    // a の次のノードに進む
71     } else {
72         w->next = b;
73         w      = w->next;
74         b      = b->next;
75     }
76 }
77
78 // 残っている要素を、末尾へ連結
79 if (a == NULL) {
80     // a が短かったということなので、余っているbのノードを繋ぐ
81     w->next = b;
82 } else {
83     w->next = a;
84 }
85
86 // 併合されたリストを返す
87 return result.next;
88 }
89
90 // リストを二分割し、再帰的にマージする
91 list_t *merge_sort(list_t *list) {
92     // 動作確認用
93     list_t *a;
94     list_t *b;
95     list_t *w; // work 変数
96
97     // 要素が0又は1であれば終了
98     if (list == NULL || list->next == NULL) {

```

```
99     return list;
100 }
101
102 // 2つのリストに分割するため、リストの中心を探す。
103 // a はリストの先頭を指し、b はリストの二番目のノードを指す。
104 // a が一つ進む間に、b は二つ進むので、
105 // b がリストの末尾に辿り着いた時には、
106 // a はリストのちょうど中央を指している。
107 a = list;
108 b = list->next;
109 if (b != NULL) {
110     b = b->next;
111 }
112 while (b != NULL) {
113     a = a->next;
114     b = b->next;
115     if (b != NULL) {
116         b = b->next;
117     }
118 }
119
120 // 動作確認用
121 // printf("----- a ----- \n");
122 // show_list(a);
123
124 // リストを前半と後半に分離する
125 // a はリスト中央を指しているので、
126 // a->next = NULL とすると list が前半部分のみになる。
127 // この時、後半の始まりが失われないよう、work変数に記憶しておく。
128 w = a->next;
129 a->next = NULL;
130
131 // 動作確認用
132 // printf("----- list ----- \n");
133 // show_list(list);
134
135 // 前半と後半のリストを、再帰的にマージする。
136 return merge(merge_sort(list), merge_sort(w));
137 }
```

付録 A

珠玉の名著のご紹介

プログラミングに関する名著のご紹介です。ご自身の血肉として頂ければ幸いです。 *¹

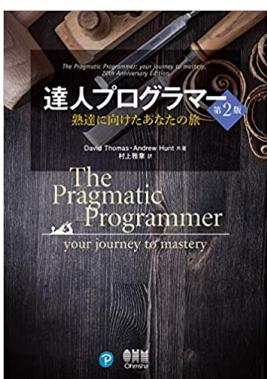
♣ 教養としてのコンピューターサイエンス講義



デジタル時代で活躍するための「教養」をこの1冊で身につけよう。
プリンストン大学の一般人向け「コンピューターサイエンス」の講義が
一冊に。デジタル社会をよりよく生きるために知識を伝説の計算機科学者
がやさしく伝えます。(著者ブライアン・カーニハン氏は、C言語の發
明者です)

本書は、わたくしたちの世界(デジタル社会)が、どのように動いてい
るのか、なぜそのしくみになっているのかをもっとも明快かつ簡潔に説
明しています。

♣ 達人プログラマー(第2版) 熟達に向けたあなたの旅



本書は、より効率的、そしてより生産的なプログラマーになりたいと願
うソフトウェア開発者に向けて、アジャイルソフトウェア開発手法の先
駆者として知られる二人により執筆されました。経験を積み、生産性を
高め、ソフトウェア開発の全体をより良く理解するための、実践的なアプ
ローチが解説されています。先見性と普遍性に富んだ本書は、入門者に
は手引きとなり、ベテランでも読み直すたびに得るものがある、座右の一
冊です。

*¹ 書籍紹介文から、引用・改変。

♣ コーディングを支える技術



本書は、プログラミング言語が持つ各種概念が「なぜ」存在するのかを解説する書籍です。世の中にはたくさんのプログラミング言語があります。そしてプログラミングに関する概念も、関数、型、スコープ、クラス、継承など、さまざまなものがあります。多くの言語で共通して使われる概念もあれば、一部の言語でしか使われない概念もあります。これらの概念は、なぜ生まれたのでしょうか。本書のテーマは、その「なぜ」を理解することです。

そのために本書では、言語設計者の視点に立ち、複数の言語を比較し、そして言語がどう変化してきたのかを解説します。いろいろな概念が「なぜ」生まれたのかを理解することで、なぜ使うべきか、いつ使うべきか、どう使うべきかを判断できるようになるでしょう。

♣ みんなのコンピュータサイエンス



コンピュータなしには生活が立ち行かなくなる水準に達しつつある現代社会。その圧倒的な力を課題解決に援用するには小手先の知識では追いつきません。とは言え無闇に全方位に知識を求めるには、その世界は広すぎ、効率も悪すぎます。

本書は計算機科学が扱う「基礎」「効率」「戦略」「データ」「アルゴリズム」「データベース」「コンピュータ」「プログラミング」という8つのジャンルにしぼり、その精髄と背景となる考え方を紹介します。

ステップアップしたいエンジニアや、ライトに全体像を俯瞰したい学生にも最適な1冊です。

♣ プログラマの数学



プログラミングに役立つ「数学的な考え方」を身につけよう。

プログラミングや数学に関心のある読者を対象に、プログラミング上達に役立つ「数学の考え方」をわかりやすく解説しています。数学的な知識を前提とせず、たくさんの図とパズルを通して、平易な文章で解き明かしています。

二進数から人工知能に至るまで、ていねいに説明しています。

プログラミングや数学に関心のある読者はいうまでもなく、プログラミング初心者や数学の苦手な人にとっても最良の一冊です。

♣ C 言語による標準アルゴリズム事典



コンピュータの算法に関するアルゴリズムの定石、レトリックを可能な限り収録した定番の書。手元に置いておきたい実用的な本が30年弱の時を経て新装改訂版として登場です。定評をいただいている基本的な内容はそのままに、時代にそぐわなくなっていた部分を改訂。これからも末長くご愛顧いただけるようにまとめ直しました。

♣ アルゴリズム図鑑 増補改訂版 絵で見てわかる33のアルゴリズム



基本的な33のアルゴリズム+7つのデータ構造をすべてイラストで解説。アルゴリズムはどんな言語でプログラムを書くにしても不可欠ですが、現場で教わることはめったになく、かといって自分で学ぶには難しいものです。

本書は、アルゴリズムを独学する人のために作りました。はじめて学ぶときにはイメージしやすく、復習するときには思い出しやすくなるよう、基本的な26のアルゴリズム+7つのデータ構造をすべてイラストで解説しています。

よいプログラムを書くために知っておかなければいけないアルゴリズムの世界を、楽しく学びましょう。

♣ C の絵本—C 言語が好きになる9つの扉



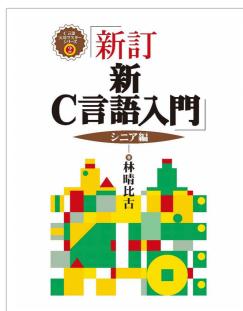
見る見るわかる！【本書の特徴】C言語には難解なトピックもあるため、文章だけではなかなかイメージがつかめず、理解しづらいものですね。本書はイラストで解説してありますので、直感的イメージがとらえられ、理解も進んでいきます。さあ、C言語への扉を開き、プログラマーへの道を進んでいきましょう！

♣ アルゴリズムの絵本-プログラミングが好きになる9つの扉



本書は、プログラミング1年生の方に向けて、プログラムを作る際のアプローチの仕方と初步的なアルゴリズムについて解説した入門書。「プログラムをいかにして組み立てて思い通りに動かすか」を重点的に解説している。特に、頭に浮かんだモヤモヤしたものをプログラムに直す際のアイデアや、ちょっと大きくて複雑なプログラムを作るとときの取り組み方について、イメージをふんだんに使って丁寧に解説している。

♣ 新・C 言語入門-シニア編-C 言語実用マスターシリーズ



本格的なプログラムへの最短コース本格的なプログラム作成に欠かせないC言語の仕様をわかりやすく解説。文法やプログラミングルールの体系的な知識の習得によって、「C言語の思想」も理解できる全プログラマー必読の1冊。困ったときのリファレンスとしても長く活用できる。

♣ C 言語ポインタ完全制覇



C言語で「難しくてよくわからない！」とつまずく人続出るのがポインタ。「Cのポインタがわからないのは、あなたが悪いわけじゃなく、単に、Cの文法がクソなだけだよ!!」第一線で活躍する筆者がCの宣言まわりの混乱した奇っ怪な文法を解き明かし、真のポインタの使い方を教授します。ポインタのみならずCへの理解が一層深まる一冊です。

♣ 小一時間でゲームをつくる 7つの定番ゲームのプログラミングを体験



書いて即実行! シンプルに積み上げていくわくわく感。C言語によるコンソールアプリで、ゲームを一手ずつ作成する手順を解説する、画期的な本の登場です。0から完成まで、手順通りに進めれば必ず完成する仕組みになっています。最小限の手順ごとに動作確認を行うので、それぞれの処理の意味を実感しながら、少しずつできあがっていく過程を楽しめます。

さあ、ゲームプログラミングの旅に出て、難しいクエストも1つずつクリアしていき、夢と冒險に溢れた未知の世界を征服していきましょう!

♣ リーダブルコード より良いコードを書くためのシンプルで実践的なテクニック



コードは理解しやすくなければならない。本書はこの原則を日々のコーディングの様々な場面に当てはめる方法を紹介する。名前の付け方、コメントの書き方など表面上の改善について。コードを動かすための制御フロー、論理式、変数などループとロジックについて。またコードを再構成するための方法。さらにテストの書き方などについて、楽しいイラストと共に説明する。日本語版ではRubyやgroongaのコミッタとしても著名な須藤功平氏による解説を収録。

付録 B

C言語 簡易まとめ

C言語の文法に関する簡易なまとめです。

♣ コメント

プログラミング言語では、ソースコード中に記述されるがコードとしては解釈されない、人に向けた文字列をコメントといいます。主にコードの記述者が別の開発者などにコードの意味や動作、使い方、注意点等について注釈や説明を加える為に使われます。^{*1}

C言語では、コメントは以下のように記述します。

記述例	説明
<code>/* コメント */</code>	複数行コメント
<code>// コメント</code>	一行コメント（便利なので多用されます）

♣ データ型

変数とは、コンピュータプログラムのソースコードなどで、データを一時的に記憶しておくための領域に固有の名前を付けたもの。^{*2}

C言語では、文字型の変数を宣言する際には、`char c;`、整数型の変数を宣言する際には、`int n;`などの型が用意されています。

コード例	説明
<code>char</code>	文字型。 一文字分のアルファベット (1 バイト 2^8 -128～+127までの整数)
<code>int</code>	整数型。 4 バイト 2^{32} -2147483648～+2147483647までの整数)
<code>long</code>	整数型。 8 バイト 2^{64} -9223372036854775808～+9223372036854775807までの整数)
<code>double</code>	倍精度浮動小数点型。 8 バイト 1.7E ± 308(有効 15 衔)までの浮動小数点数)

♣ リテラル

^{*1} 出典：IT 用語辞典

リテラル (literal) とは、直値、直定数とも呼ばれ、コンピュータプログラムのソースコードなどの中に、特定のデータ型の値を直に記載したものである。また、そのように値をコードに書き入れるために定められている書式のことをいう。^{*3}

コード例	説明
123	10進数の整数リテラル
0x30A2	16進数の整数リテラル

♣ 文字列

文字列とは、文字を並べたもの。コンピュータ上では、数値など他の形式のデータと区別して、文字の並びを表すデータを文字列という。^{*4}

C言語には、「文字列」型は用意されていないため、「文字」の「配列」として表します。

コード例	説明
char princess[20] = "shirayukihime";	ダブルクオートの文字列リテラル。
printf("%c", princess[0]);	「s」が出力される。
printf("%s", princess);	「shirayukihime」が出力される。

♣ 演算子

演算子とは、数学やプログラミングなどで式を記述する際に用いる、演算内容を表す記号などのこと。様々な演算子が定義されており、これを組み合わせて式や命令文を構成する。^{*5}

以下の表は優先順位の最も高いものから最も低いものの順に並べられている。^{*6}

*6 出典: 演算子優先順位と結合順序 (<https://www.ibm.com/docs/ja/i/7.5?topic=operators-operator-precedence-associativity>)

コード例	説明
. または->	メンバー選択
[]	添え字
()	関数呼び出し
++	後置増分
--	後置減分
sizeof	サイズ
++	前置増分
--	前置減分
~	ビット単位否定 (1 の補数)
!	否定
-	単項減算
+	単項正
&	アドレス取得
*	間接参照
()	型変換 (キャスト)
*	乗算
/	除算
%	剰余
+	二項加算
-	二項減算
<<	左シフト
>>	右シフト
<	小なり
<=	以下
>	大なり
>=	以上
==	等価
!=	不等価
&	ビット論理積
^	ビット排他的論理和
	ビット論理和
&&	論理積
	論理和
? :	条件式・三項演算子
=	単純代入
*=	乗算代入
/=	除算代入
%=	剰余代入
+=	加算代入
-=	減算代入
<<=	左シフト代入
>>=	右シフト代入
&=	ビット積代入
=	ビット和代入
,	カンマ

♣ 制御構造

プログラムの流れを制御するための構文です。繰り返しのための「**for 文**」、条件分岐のための「**if 文**」などが用意されています。

例	説明
<code>while(x){}</code>	while ループ。 x が true なら反復処理を行う。 繰り返し回数が不明な際に用いると効果的
<code>for(x=0;x < y ;x++){}</code>	for ループ。 x < y が true なら反復処理を行う。 繰り返し回数が分かる時に使うと効果的
<code>if(x){/*A*/*}else{/*B*/*}</code>	条件式。 x が true なら A の処理を、 それ以外なら B の処理を行う
<code>switch(x){case "A":{/*A*/*} "B":{/*B*/*}}</code>	switch 文。 "A" なら A の処理を、 "B" なら B の処理を行う
<code>x ? A: B</code>	条件（三項）演算子。 x が true なら A の処理を、 それ以外なら B の処理を行う
<code>break</code>	break 文。 現在の反復処理を終了しループから抜け出す。
<code>continue</code>	continue 文。 現在の反復処理を終了し次のループに行く。

♣ データアクセス

プログラミング言語 Pascal の開発者 ニクラウス・ヴィルト氏による、「プログラミング」 = 「データ構造」 + 「アルゴリズム」 は、広く知られています。

配列という主要なデータ構造にアクセスするために、次の構文が用意されています。

コード例	説明
<code>array[0]</code>	配列へのインデックスアクセス

♣ 関数宣言

関数とは、コンピュータプログラム上で定義されるサブルーチンの一種で、数学の関数のように与えられた値（引数）を元に何らかの計算や処理を行い、結果を呼び出し元に返すもののこと。

*7

サンプル	説明
<code>int add(x, y){ return x + y; }</code>	関数の一例 仮引数 x と y の和を返す関数

終わりに

本書では、★1のこんにちは世界から始まり、★5のポーカーや双六、マージソートまで、様々なコードを作成いたしました。

全部で、50本のプログラム、自分で作ったプログラムの体験はいかがでしょうか。ぜひ、遊んでみてください。そして、いろいろ創意工夫して、より発展させたプログラムを作って見られてください。

「福祉」。「福」「祉^{*8}」どちらも「めぐみ、さいわい」という意味を持ちます。

「熱き心、^{たくま}逞^{かいな}しき腕、冷静な頭脳」

学生時代に言われた言葉ですが、福祉を生きる者は、人としての熱い思い、暖かい心を持ち、その上で、冷静な判断力を以て、力強く行動するのだと。

「工学」の「工」は、「天の^{ことわり}理^り」を、地に下ろす」意味です。

技術の産物としての社会ではなく、世界を^{かがや}耀^{かがや}かせるために技術を用いてください。技術に使われるのではなく、技術を使いこなし、人の道に役立てる人となってください。

令和の御世を生きる皆さんに素晴らしい人生を生き、素晴らしい日本を創ることを願って筆を置きます。

いやさか
彌榮

*8 「祉い」と書いて、「さいわい」と読みます。天からの恵みがその身に止まる意味です。

はじめてのC言語 練習帳

令和五年十一月十一日 ver 4.0.0

著 者 アトリエ未来

発行者 早乙女 遙香

連絡先 contact@atelier-mirai.net

<https://atelier-mirai.net>

© 令和五年 アトリエ未来