

# C 言語 基礎講座

- C言語を学ぶ方へ -



---

# Table of Contents

Introduction	1.1
環境構築	1.2
C言語とは	1.3
データ型	1.4
書式指定子	1.5
コラム	1.6
コンピュータを創った人々	1.6.1
コンピュータの仕組み1	1.6.2
コンピュータの仕組み2	1.6.3
コンピュータ5大機能と動作原理	1.6.4
プログラム開発の手順	1.6.5
アルゴリズム+データ構造	1.6.6
単位のお話	1.6.7
画素の話	1.6.8
タッチタイピング	1.6.9
ショートカットキー	1.6.10
Unix コマンド	1.6.11
学習に役立つサイトのご紹介	1.7
学習に役立つ本のご紹介	1.8
資格について	1.9
講座	1.10
初めてのプログラム	1.10.1
繰り返し処理	1.10.2
条件分岐・配列	1.10.3
ファイルの取り扱い・文字の検索	1.10.4
ポインタ	1.10.5
棒取りゲーム	1.10.6
構造体	1.10.7
マージソート	1.10.8
在庫管理システム	1.10.9

---

データベース	1.10.10
補足 その1	1.10.11
補足 その2	1.10.12
オブジェクト指向	1.10.13
練習課題	1.11
難易度 ★☆☆☆☆☆☆☆	1.11.1
難易度 ★★★☆☆☆☆☆	1.11.2
難易度 ★★★★☆☆☆☆	1.11.3
難易度 ★★★★★☆☆☆	1.11.4
難易度 ★★★★★☆☆☆	1.11.5
難易度 ★★★★★★★☆	1.11.6
難易度 ★★★★★★★★	1.11.7
解答例	1.12
難易度 ★☆☆☆☆☆☆☆	1.12.1
難易度 ★★★☆☆☆☆☆	1.12.2
難易度 ★★★★☆☆☆☆	1.12.3
難易度 ★★★★★☆☆☆	1.12.4
難易度 ★★★★★★★☆	1.12.5
難易度 ★★★★★★★★	1.12.6
おまけ	1.12.7

# C言語 基礎講座

- C言語を学ぶ方へ -

## 環境構築

### 皆様、C言語講座へようこそ！

#### 環境とは？

プログラミング環境には、利用するツールから分類した場合

1. テキストエディタ＋ターミナル
2. 統合開発環境(IDE)

の二つに分類されます。

また、

1. ローカル環境
2. クラウド環境

の二つの観点から分類することも出来ます。

ここでは、

1. テキストエディタ＋ターミナル
2. ローカル環境

の組み合わせで、開発環境を構築していきます。

---

それでは初めていきましょう！

#### 用意するもの

- Mac (または PC)

プログラミングを *Mac* で行うと、いろいろな開発ツールが豊富に提供されており、快適にコーディングを行うことができます。特段の理由がない限り、*Mac* がお薦めです。

また、広いディスプレイは、エディタやターミナルを開きつつ、調べ物のためにブラウザを開くなど、とても効率が上がります。

究極のデスクトップ体験。圧倒的な、美しい進化を遂げた、27インチiMac Retina 5Kディスプレイモデル

- キーボード

アメリカ西部のカウボーイたちは、馬が死ぬと馬はそこに残していくが、どんなに砂漠を歩こうとも、鞍は自分で担いで往く。馬は消耗品であり、鞍は自分の体に馴染んだインターフェースだからだ。いまやパソコンは消耗品であり、キーボードは大切な、生涯使えるインターフェースであることを忘れてはいけない。【東京大学 名誉教授 和田英一】

いっさいの妥協を許さず、上質のこだわりを形にした最上級クラスの小型キーボード  
[Happy Hacking Keyboard](#)

- トラックパッド

GUI操作の為のマウスも良いですが、トラックパッドはもっと快適です。あなたのお気に入りのコンテンツをこれまで以上に快適にスクロールしたりスワイプできるようになります。感圧タッチテクノロジーも加わったので、強めに押すと様々な操作ができ、一つのクリックからより多くのことを引き出せます。

[Magic Trackpad 2](#)

- 教科書
- ノート
- 筆記用具
- USBメモリ / (またはファイル保存可能なクラウドサービス)

## 環境構築

C言語専用プログラミング環境を作ろう！このページに沿って、作業開始して下さい。

[WindowsにAtomでお手軽にC言語環境を作る方法](#)

1. 多言語対応テキストエディタ：ATOM入手しよう。

ATOMとは、GitHub社により提供されているオープンソースソフトウェア(OSS)です。無料でありながら、とっても高機能。プログラマ御用達のテキストエディタです。

一般に文章を作るときには、一太郎やPages, LibreOffice, MS-Wordのようなワープロソフトを使って作成します。

プログラムを作る際には、テキストエディタ（エディタ）と呼ばれる、プログラム作成用のソフトウェアを用います。

ダウンロードサイト：<https://atom.io/>

ATOMのダウンロード方法: <http://webkaru.net/dev/windows-atom-install/>

各ツールの保存を待っている間に下記をやりましょう!

このサイトを読みましょう!

ATOMの日本語化: <http://psycho-phantom.com/atom-windows-1403>

プログラムでは一般に英語が用いられています。また用いるツールも海外で開発されたものが多いことから、メニュー表示などが英語になっています。簡単な英語は読めるようになっておいた方がよいとは思いますが、好みに応じて日本語化することも可能です。

ATOMパッケージ: <http://qiita.com/cw-shibuya/items/02726b2dcef015ee65bf>

- パッケージとは、ATOMの拡張機能のことです。ATOMは標準でも十分に快適に使えるように作られていますが、たくさんの有志の方が、様々な便利な拡張機能を提供して下さっています。
- ATOMの作業をより効率化するためのパッケージについては、各自でお好みで入れることが出来ます。ただし、多数のパッケージを導入した結果、パッケージ間の相性などによりATOM自体が使えなくなる場合もあるので、適宜ご利用下さい。
- linter-gcc はC言語の文法上の誤りを指摘するパッケージ（拡張機能）です。ATOMパッケージの linter-gcc をインストールするとエラーメッセージが表示されます。コンパイラ(gcc)「gcc.exeのあるファイルがどこか分からないので、設定してください」というお知らせです。gcc.exeのあるファイル場所をご確認の上で設定してください。

ATOMのショートカットキー: <http://pasolavo.com/web/atom-keybinding-for-windows.html>

プログラム開発に当たっては、キーボードから入力していきます。慣れない間はマウス操作しても良いですが、習熟するにつれ、いちいちマウスに手を動かすと作業効率がとっても落ちます。

快適にプログラミングを楽しむためにも、ショートカットキーをマスターしましょう。

## 2. C言語ファイルを入れるための作業用フォルダを作成します。

フォルダ名は、任意ですが、ここでは説明の都合上 work とします。

## 3. C言語の学習ポイント C言語学習のための概要が簡潔に纏められています。初学者にお薦めです。 <http://www9.plala.or.jp/sgwr-t/>

## 4. C言語関数リファレンス C言語には様々な関数（機能のまとめ）が用意されています。これらの関数を習得することがC言語マスターへの第一歩です。

<http://orchid.co.jp/computer/cschoo/cref.html>

## 5. ターミナルの設定

Macには、標準で ターミナルと呼ばれるアプリケーションが付属しています。これでも十分ですが、iTerm2 (<https://www.iterm2.com>) が非常に優れていますので、こちらの利用をお薦めいたします。

## 6. コマンドプロンプトの設定

Windows 環境の方への説明です。

Windowsには、コマンドプロンプトと呼ばれるCUI操作が行えるプログラムが付属しています。(CUIとは、コマンドをタイプすることで、様々な作業をコンピュータに行わせられる操作体系のことです。)

コマンドプロンプトはよく使いますので、デスクトップ等にショートカットを作成しておくと便利です。

(コマンドプロンプトには、UTF-8 で日本語入力が出来ないなどの支障があります。これを解消するために ComEmu を導入するのも一つの方法です。)

## 7. gcc のインストール

gcc は C 言語のプログラムをコンパイル（翻訳）するためのソフトです。

コンピュータは、人に分かるように書かれたプログラミング言語（ソースコードといいます）を理解することが出来ません。ですので、コンピュータに分かる言語（機械語）にコンパイル（翻訳）する必要があります。

いくつものコンパイラがありますが、ここではフリーで広く使われている *gcc* をインストールします。

## 8. Mac の方

- 開発環境をインストールするために homebrew というソフトが広く使われています。  
"homebrew gcc インストール" で検索して下さい。
- homebrew は さまざまなツールをインストールすることが出来ますが、慣れない間は少し難しいかも知れません。 "mac gcc インストール" で検索すると、「MacでC言語 - コンパイラ (gcc) のインストール - Xcode - Command Line Tools」  
<http://webkaru.net/clang/mac-compiler-gcc-install/>

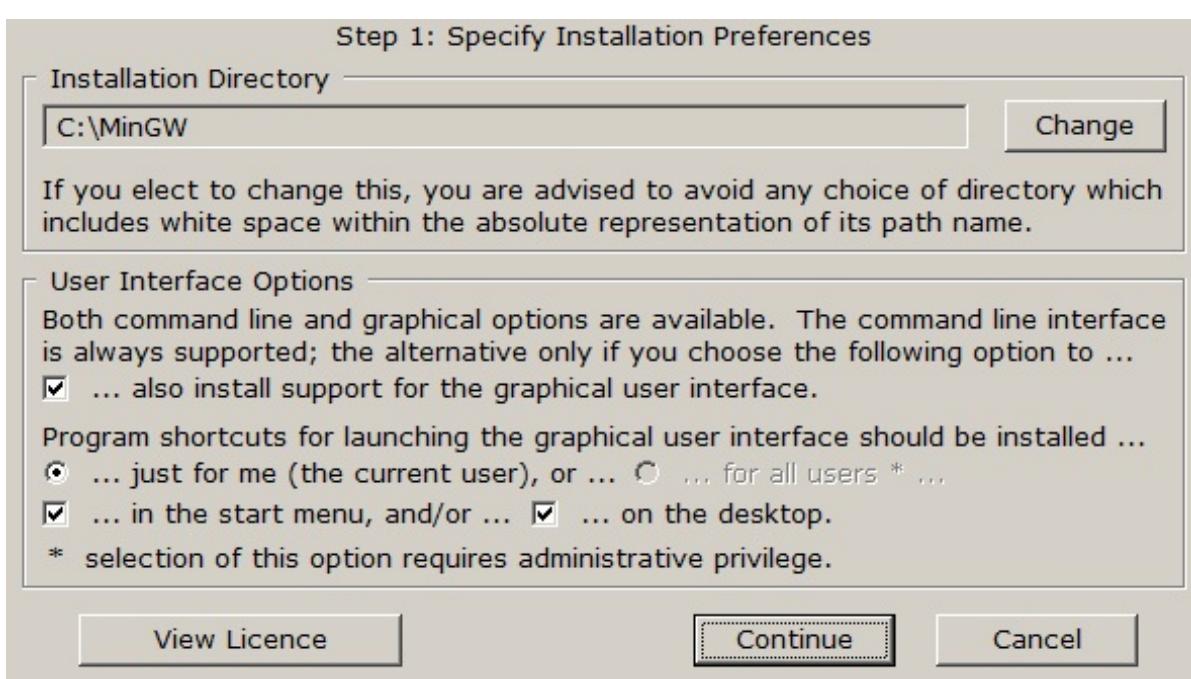
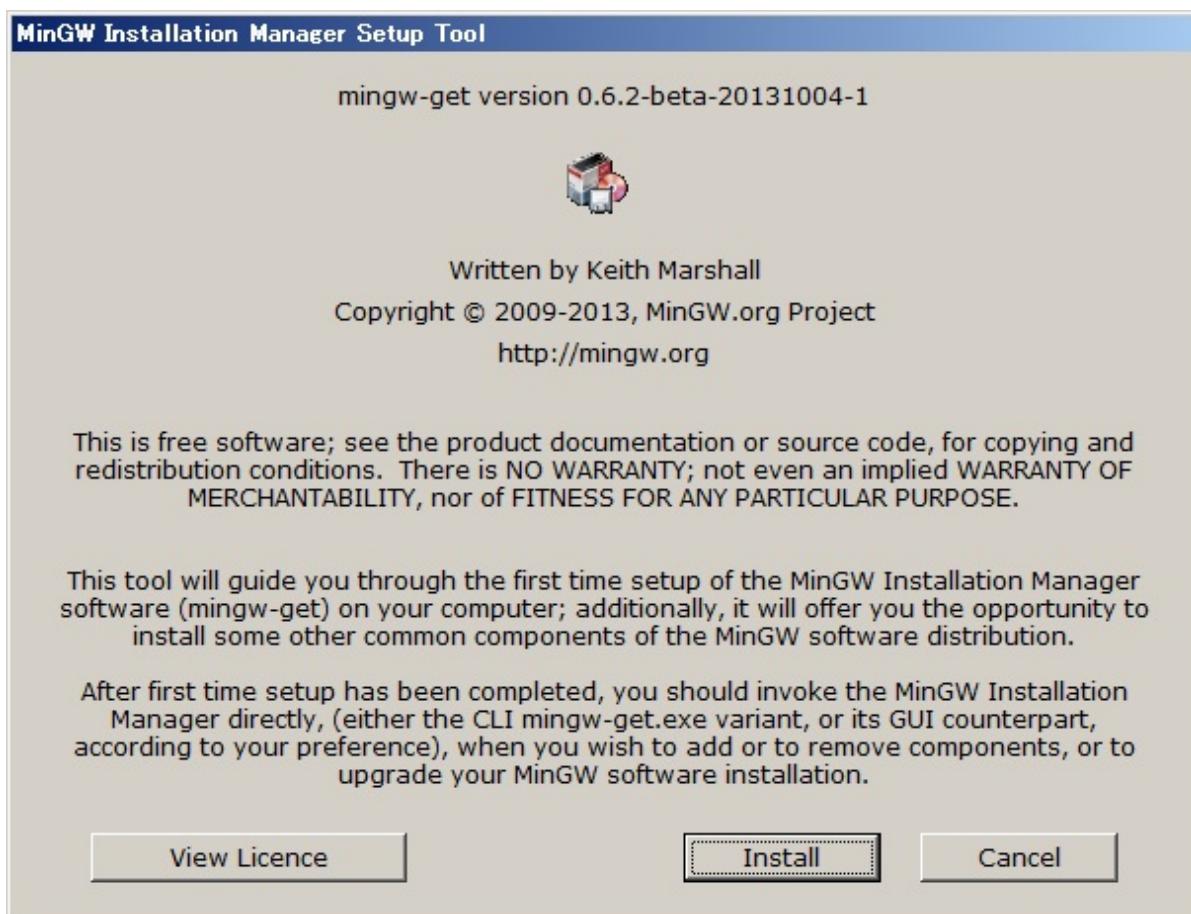
いずれかの方法で、gcc をインストールして下さい。

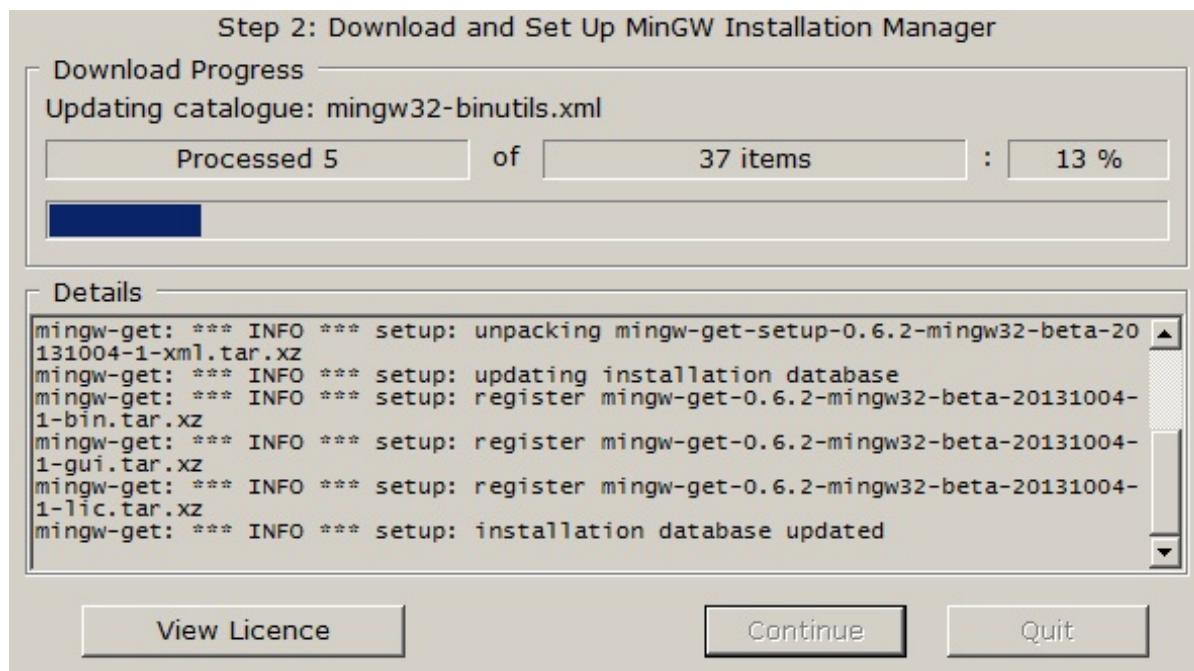
## 9. Windowsの方

gcc を Windows 対応させた *MinGW* を使います。

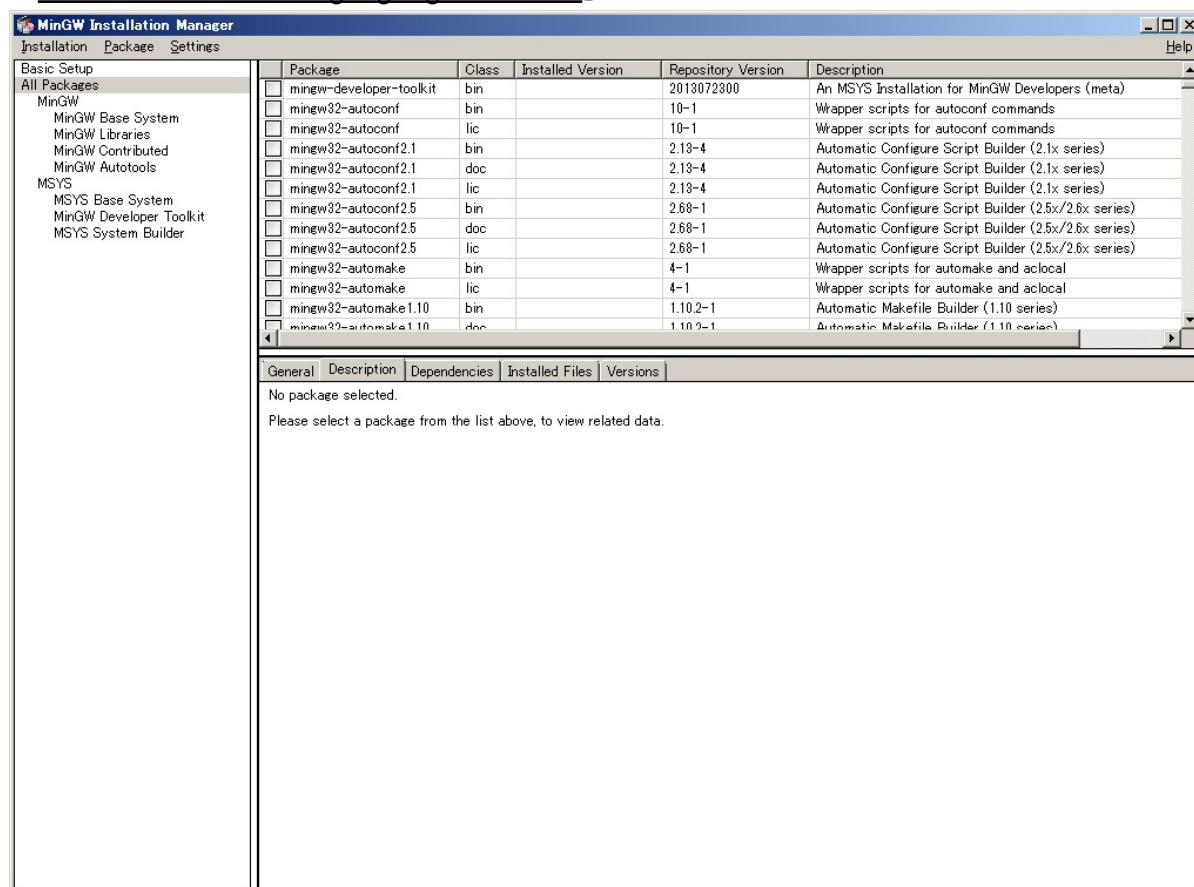
WindowsでのC言語の開発環境（Eclipse）<http://gabekore.org/windows-c-eclipse> より、引用

- MinGWのWebサイト(<http://www.mingw.org/>)で右上の「Download Installer」ボタンをクリック。[sourceforge](http://sourceforge.net)というサイトに飛びます。
- 自動でダウンロードが始まります。(始まらなければ「Download mingw-get-setup.exe」をクリック)
- ダウンロードした mingw-get-setup.exe ファイルを起動
- インストール自体は「Install」ボタンや「Continue」ボタンをクリックすると完了します。

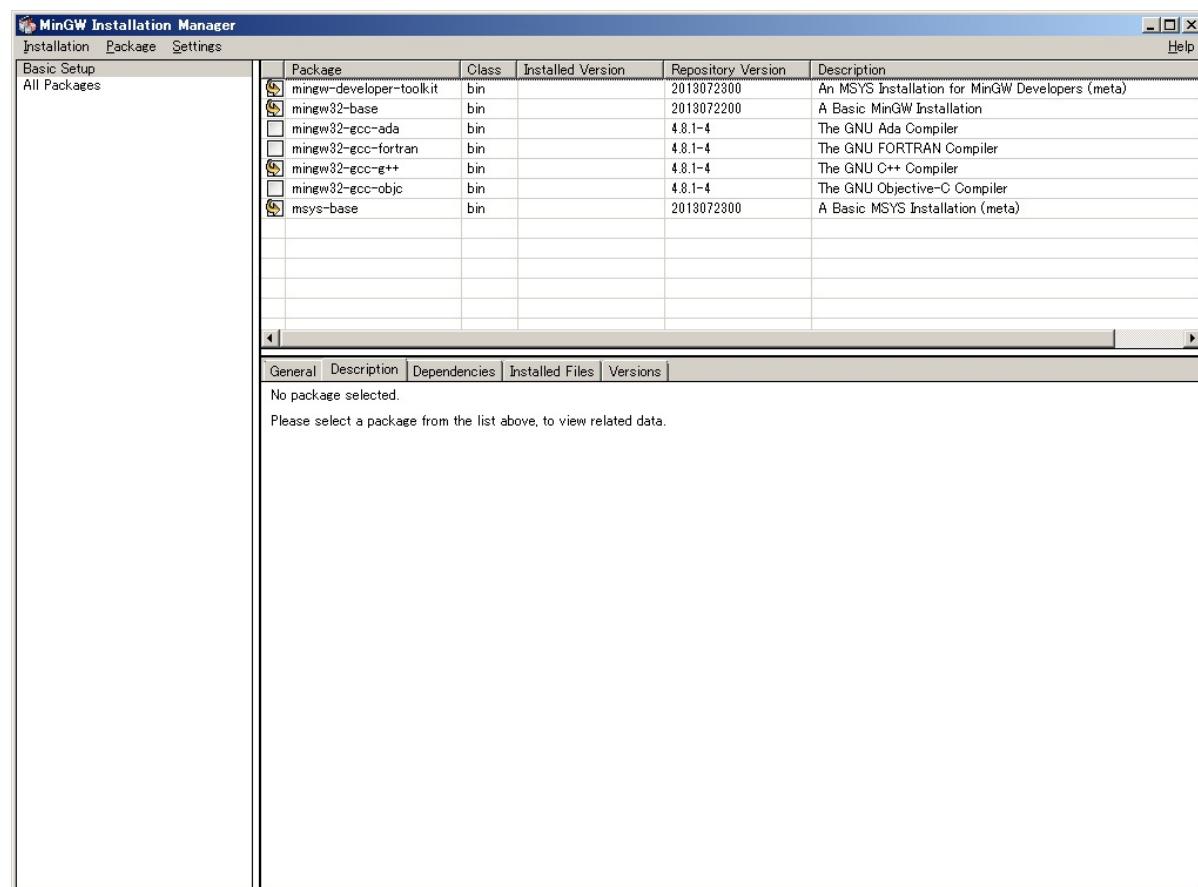




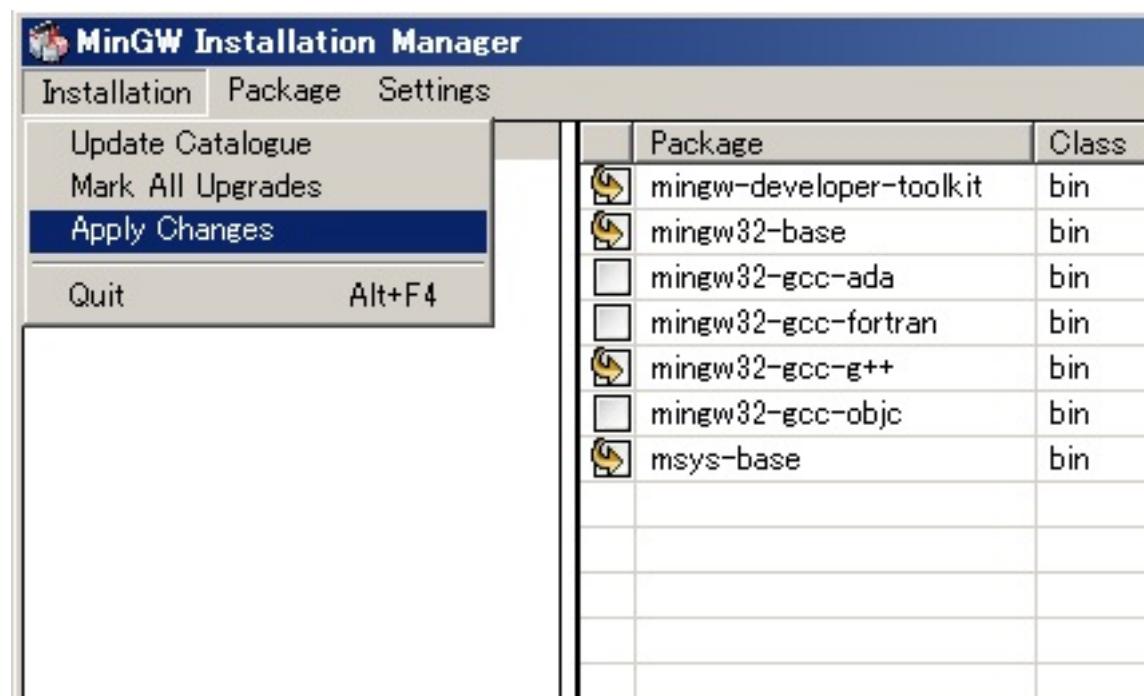
5. 以下の様な画面（MinGW Installation Manager）が出てきます。 出でこなければ  
「C:\MinGW\libexec\mingw-get\guimain.exe」を起動

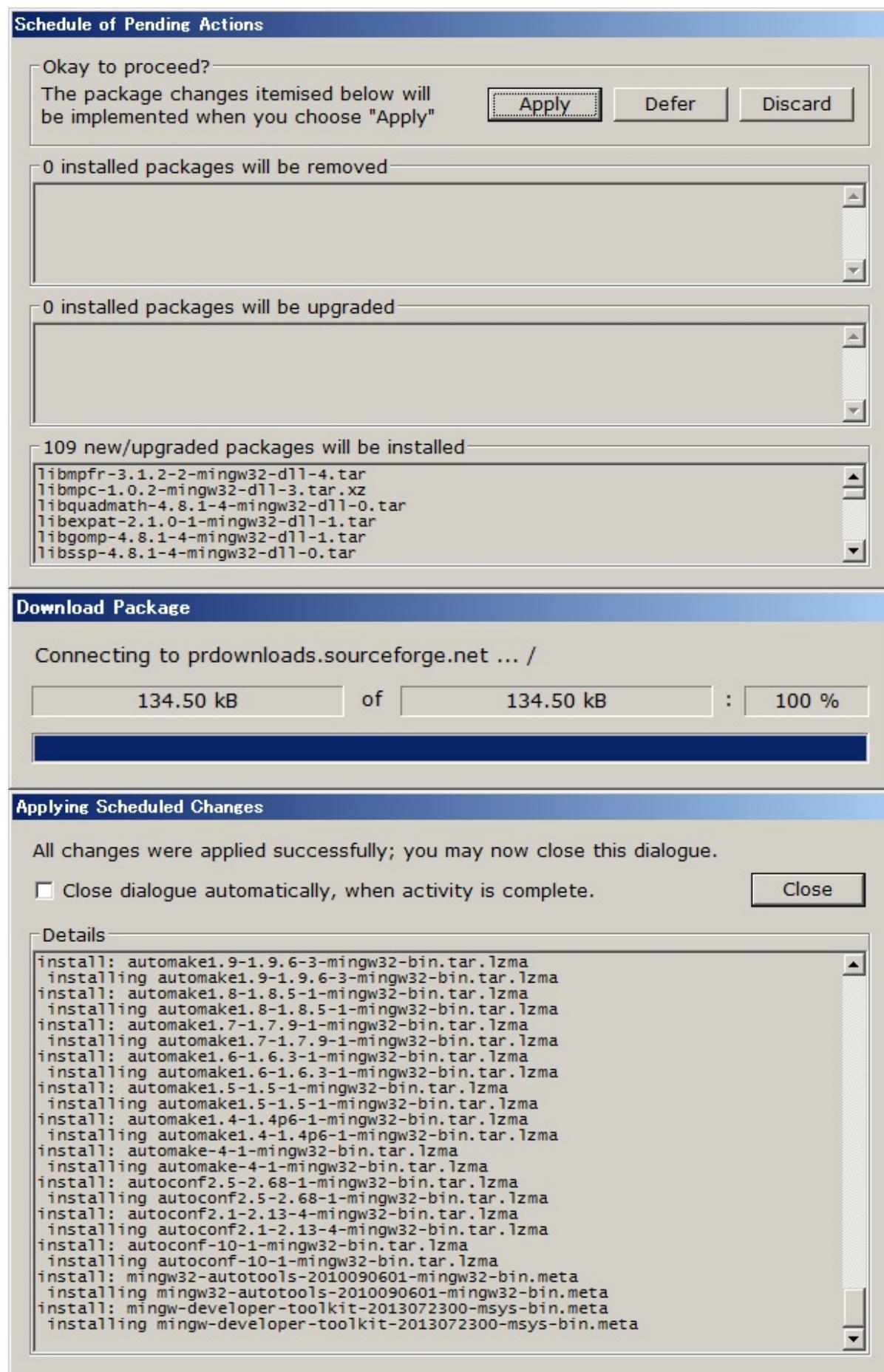


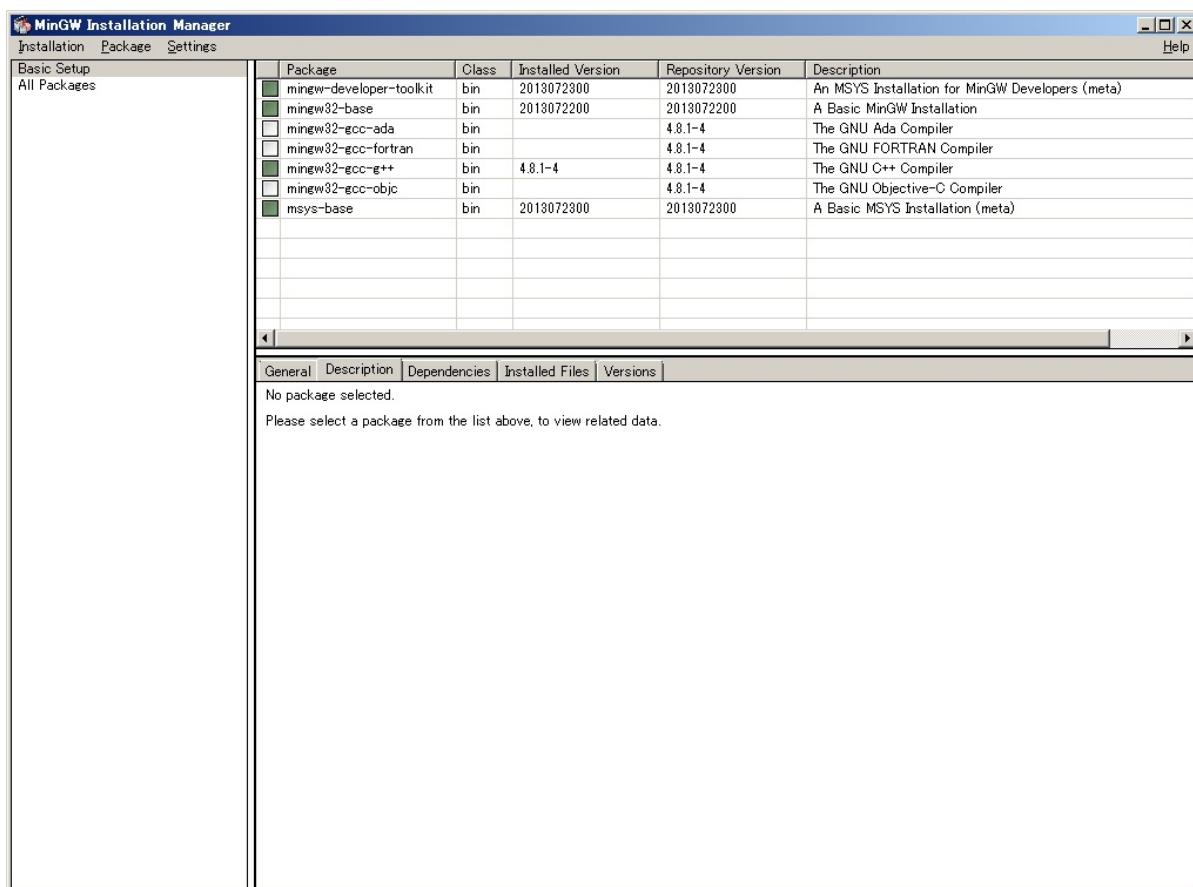
6. C言語をコンパイルするだけであれば、以下の2つを選択（左ペインで「Basic Setup」を選択）
- mingw32-base
  - mingw32-gcc-g++



7. 選択が終われば [Installation] - [Apply Changes] でインストール開始







- 以上で、MinGWのインストールは完了です。

## 設定

MinGWを使うためには、「C:\MinGW\bin」に、PATHを通す必要があります。そうすると、コマンドプロンプトから gcc でコンパイルできます。PATHの通し方は、次のとおりです。

- エクスプローラのコンピューターで右クリックしてプロパティ選択
- 左側のメニューで「システムの詳細設定」をクリック
- 「詳細設定」タブクリック
- 「環境変数(N)」クリック
- 「システム環境変数(S)」エリアの、「Path」をダブルクリックor「編集(I)」をクリック(無かったら新規作成してください)
- 「システム変数の編集」ダイアログの「変数値(V):」の一番後ろに「;C:\MinGW\bin」を追記
- 「OK」ボタンクリック

## 使い方

- 以下の内容でCファイルを作成

```
#include  
main(){  
    printf("Hello, World\n");  
}
```

2. 例えば「hello.c」という名前で「c:\work」フォルダに保存
3. コマンドプロンプトを起動。  
(Windowsキー + R を押す。cmdと入力。Enterキーを押す)
4. コマンドプロンプト(黒い画面)が立ち上がる
5. 「cd c:\work」を入力+Enterキー押下
6. 「gcc hello.c」を入力+Enterキー押下
7. 「c:\work」にa.exeが作成されます
8. 「a」を入力+Enterキー押下
9. 「Hello, World」と表示されます

# C言語とは

(Wikipediaより)

C言語（シーゲンゴ）は、1972年にAT&Tベル研究所のデニス・リッチーが主体となって開発したプログラミング言語である。英語圏では単に C と呼んでおり、日本でも文書や文脈によっては同様に C と呼ぶことがある。

## 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。
- 機械語に変換するソフトなどの開発環境がCPUに付属していたり無償だったりするものもあるため、ライセンス料の支払いをしなくても使用が始められる。
- 開発時期が古く、文法に機械語の影響が強く、複雑である。この欠点を補正するためのちに開発された新言語に比較し、記述が多く、面倒で習得しにくい低級言語である。(IT用語における低水準、低級、低レベルという表現は、機械語やアセンブリ言語のように、ハードウェアの動作をより直接的に記述できる（あるいはしなければならない）ことを指す。C言語に関しては、ポインタを通じて直接的にメモリ操作を行うことができる点などで、Javaなどに比べて低級であると言える（反対にJavaはC言語に対して高レベルな言語である）。通常、低級言語と高級言語は動作環境や生産性などに応じて使い分けられている。低級、高級という表現は誤解を招きがちだが、プログラミング言語としての質を指す言葉ではないことには注意が必要である。)
- 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。
- 言語の適用先であるUNIXの場合、大抵のことがスクリプト言語・マクロプロセッサやフィルタやそれらの組み合わせで処理できるため、うまく分野の棲み分けができていた面があった。仕様規格・派生言語も多く幅広い領域への移植の結果、適切でない分野にC言語が使われている場合もある。
- C言語は手続き型言語の側面と関数型言語の側面がある。コンパイラ言語とOSを念頭に設計している。アセンブリ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。低水準な記述が出来る高級言語とも、高級言語の顔をした低級言語と言ふことがある。

- 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。セキュリティーの脆弱性や潜んだ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がある。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。自動車分野ではMISRA-CというC言語部分集合（subset）を定義して、C言語の弱点を補っている。
- 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。
  - 記述作法について、しばしば議論の対象となり、本も多数出版。
- 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。原理的に無条件分岐（goto文）を使用する必要はなく、MISRA-Cでは当初goto文を禁止していた。goto文を使わなければ、スペティプログラムと呼ばれる読みにくいプログラムになりにくい。
- 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。なお、C言語ではUNIXのようなOSを前提としたホスト環境と、割り込み制御のようなOSを前提としないフリースタンディング環境とがある。ホスト環境では、プログラム開始直後に実行するプログラム要素をmainという名前の関数として定義する。フリースタンディング環境では、エントリポイントと呼ばれるアドレスに置かれたコードをプログラムの開始点とするが、それがmain関数である必要はない。なお再帰呼び出しは、スタックオーバフローの原因となるため、MISRA-Cでは禁止している。
- 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。MISRA-Cのようないくつかの制約では、C言語規格が規定している関数の妥当性について指摘し、いくつかの関数を利用しないように規定している。
- 配列とポインタとでは、宣言の仕方と領域確保の仕組みは異なっているが、配列アクセスそのものは、ポインタ演算および間接参照と同等である。ポインタを配列表記でアクセス

- することや、配列をポインタ表記でアクセスすることができる糖衣構文がある。MISRA-Cでは、応用ソフトを記述することが目的であるため、ポインタの利用を禁止している。
- ソースコードの記述に使う文字集合はANSI-C:1989(ISO/IEC 9899:1990)ではASCIIを標準としている。他のISO 646でも書けるように、3文字利用したトライグラフと呼ばれる表記法も存在する。その後、ISO/IEC 9899:1995 AMDなどでは多バイト対応の拡張を規定している。さらに、その後トライグラフは複数のコードを利用したシステムでしか利用がないため、より分かり易い2文字によるダイグラフを規定している。

## アセンブラーとのインターフェース

- 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。アセンブラーとのインターフェースは#pragma asmなどを用いて局所化を図る努力はあるが、コンパイラごとに定義があり、CPUが同一であっても移植性が低い場合がある。

## コンパイラ仕様

- コンパイラの処理が1パスで済む仕様になっている。ANSI-C:1989では宣言のない変数はintを想定することになっていた。ISO/IEC C:1999以降では変数はその使用より前に宣言する必要がある。関数の宣言がないと、戻り値や引数をint型とみなす仕様は、自由な発想を促すプログラミングの視点で好ましいが、型検査・型証明の仕組みが十分にないと不具合の原因になることがある。後継言語では記述によって先読みが必要になりうる。
- マクロ記述やコンパイル条件の指定などが出来る前処理指令が標準化されている。前処理指令の解釈をするプリプロセッサを持っている。プリプロセッサ（preprocessor）は、その名の通りコンパイル処理の前に自動的に実行される。コンパイラの機能として、プリプロセッサを通しただけの段階のソースコードを出力可能になっているものがある。前処理の結果を検査することで、設計者の意図と前処理の結果のズレがないか確認できる。

## 処理系の簡素化

ホスト環境や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バッファオーバーフロー（固定長のバッファをはみだして上書きが行われてしまう）である。標準ライブラリにはバッファオーバーフロー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、プログラムにより制御する事で可変長配列を可能にしている。
- 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い。

- 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開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。

## 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があるが、メモリ制約などで動作しないことがある。

## コード例

### Hello world プログラム

C言語のHello world プログラムには、ホスト環境を前提とするか、フリースタンディング環境を前提とするかで、方向性が異なる。入門書によって趣が異なるいくつかの方向性が存在する。ホスト環境を前提とする場合には、入出力の利用により、動作をすぐに確かめることができる。標準Cライブラリ `stdio.h` の `printf` 関数を利用したものを例示する場合、以下のようなものがある。

```
#include <stdio.h>

int main(void){
    printf("Hello, world!\n");
    return 0;
}
```

ここでサンプルソース中の「\n」は改行を表す。なお、`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。C言語として可変引数である特殊な`main`と`printf`を使っていることにより、C言語の関数プログラミングに対する誤解を生む原因にもなっている。

### ====主な制御構造=====

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

## 歴史

## 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った。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が多く、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パイ

ラ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。

## C言語の規格

### K&R

リッチ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。

### C89/C90

そこで、国際標準化機構(ISO)/IEC JTC1と米国国家規格協会(ANSI)は協同でC言語の規格の標準化を進め、1989年12月に ANSI が ANSI X3.159-1989, American National Standard for Information Systems -Programming Language-C を、1990年12月に ISO が INTERNATIONAL STANDARD ISO/IEC 9899 : 1990(E) Programming Languages-C を発行した。ISO/IEC 規格のほうが章立てを追加しており、その後 ANSI も ISO/IEC 規格にならって章立てを追加した。それぞれ C89 (ANSI C89) 及び ISO/IEC C90 という通称で呼ぶことがある。日本では、これを翻訳したものを日本工業規格『JIS X3010-1993 プログラム言語C』として、1993年10月に制定した。

最大の特徴は、C++と同様の関数プロトタイプを導入して引数の型チェックを強化したことと、`void` や `enum` などの新しい型を導入したことである。一方、処理系に依存するところに留めた部分も幾つかある（`int` 型のビット幅、`char` 型の符号、ビットフィールドのエンディアン、シフト演算の挙動、構造体などへのパディング、等）。

16bit/32bitCPUの両方に対応できるようにするために、定義しないことを決めている未定義、定義したものとのどにするかを決めていない未規定、処理系ごとに決めて文書化する処理系定義など、CPUとの相性による有利不利が生じないような規定になっている型の大きさは16bit/32bitCPUの両方に対応できるように決めている。バイト数は `sizeof` 演算子で取得し、

最大最小値は `limits.h` で参照することとしている。多くの処理系では `char` 型は 8 ビット、`short` 型は 16 ビットである。`int` や `long` は CPU のレジスタの幅などによって決めている。また API などの呼び出しには、ヘッダで `BYTE` や `WORD` などと `typedef` で定義した型を使用して回避することがある。`char` 型以外で符号を明示しない場合は `signed` (符号付き) にするかどうかも処理系。

## C99

1999年12月1日に、ISO/IEC JTC1 SC22 WG14 で規格の改定を行い、C++ の機能のいくつかを取り込むことを含め機能を拡張し ISO/IEC 9899:1999(E) Programming Language--C (Second Edition) を制定した。この版の C 言語の規格を、通称として C99 と呼ぶ。日本では、日本工業規格 JIS X 3010:2003 「プログラム言語C」 がある。

日本では、日本工業規格 JIS X 3010:2003 「プログラム言語C」 がある。

主な追加機能：

- 変数宣言がブロックの先頭でなくても良くなった。
- ブール代数を扱うための `_Bool` 型が予約語に追加され、標準ライブラリーとして `stdbool.h` を追加した。
- 複素数を扱うための `_Complex` 型や `_Imaginary` 型を予約語に追加し、標準ライブラリーとして `complex.h` を追加した。
- 64 ビット整数値を保持できる `long long int` 型の追加。
- `//` による 1 行コメント。
- インライン関数 (`inline` キーワード)。
- 可変長配列 (`alloca` 関数の代替)

## C11

2011年12月8日に改定された ISO/IEC 9899:2011 (通称 C11) が現在の最新規格である。gcc や Clang などが部分的に対応している。改定による変更・追加・削除機能の一部を以下に記述する。

C11 は Unicode 文字列 (UTF-32、UTF-16、UTF-8 の各符号化方式) に標準で対応している。そのほか、`type-generic` 式、C++ と同様の無名構造体・無名共用体、排他的アクセスによるファイルオープン方法、`quick_exit` などのいくつかの標準関数などを追加した。

`gets` 関数は廃止

## 主なC言語処理系

gcc, clang, Visual C++, C++Builder など著名な 4 つが C 言語と C++ を一つの処理系で対応している。C 言語と C++ の共通部分を明確にし、二つの言語の違いに矛盾が生じないようにすることが課題になっている。

## Linux、Windows、UNIX用

- GNUコンパイラコレクション（GCC） C・C++ 以外の言語もサポートし、多数のCPUやオペレーティングシステムに対応、組み込み向けも含む多様な開発に広く使われるオープンソースのコンパイラ。
- clang LLVM をバックエンドとして用いるオープンソースのC言語・C++・Objective-C コンパイラ。多数のCPUに対応。
- Microsoft Visual C++ Windows系プラットフォーム用のC言語・C++コンパイラ。2013からC99におおむね対応。x86・x64が主だがXbox 360、Windows CE等向けにPowerPC、ARM、MIPS、Itanium等に対応した版もある。前身としてMS-DOS・Windows用のMicrosoft C Compilerがある。またその廉価版としてQuick Cがあった。
- C++ Builder Windows用のx86用C言語・C++のコンパイラ。RAD。前身はDOS、Windows用のBorland C/C++。さらに前身としてTurbo C/C++がある。

## 組み込み用、8ビット・16ビット・32ビット・64ビットCPU用（クロスコンパイラ）

- CodeWarrior C/C++ 組み込み向けやゲーム機開発向けのC言語・C++コンパイラ。Mac OS用として発祥、かつてはWindows用・BeOS用・Palm用もあった。
- ARM C/C++ armCPU用C言語・C++コンパイラ。
- High C 元はx86向けでPC/AT互換機用だが80386のネイティブモードに対応したためFM TOWNSでも標準開発環境、「High C 386」として使用された。現在は各社RISC向け。
- BDS-C CP/M(8080・Z80)用のサブセット(整数のみ)のK&R系のC言語コンパイラ。現在はパブリックドメインソフトウェア。
- Hitech-C Z80、PICなど。
- Lattice C 1980年代に、日本で高い普及率を見せたコンパイラ。解説書も多く出版されていた。日本での発売はライフポート。初期版はマイクロソフトCコンパイラ1.0として発売された。商用利用のできない個人向けの「personal」版も販売されており、この価格は19,800円であった。
- LSI C 8080・Z80用のLSI C-80(セルフ版・クロス版。現在はクロス版のみ)と、8086用のLSI C-86がある。8086では機能限定(スマートモデルのプログラムしか開発できず、デバッガがない)の「試食版」がフリーソフトで公開され、広く使われた。

## 関連するプログラミング言語

### 先祖

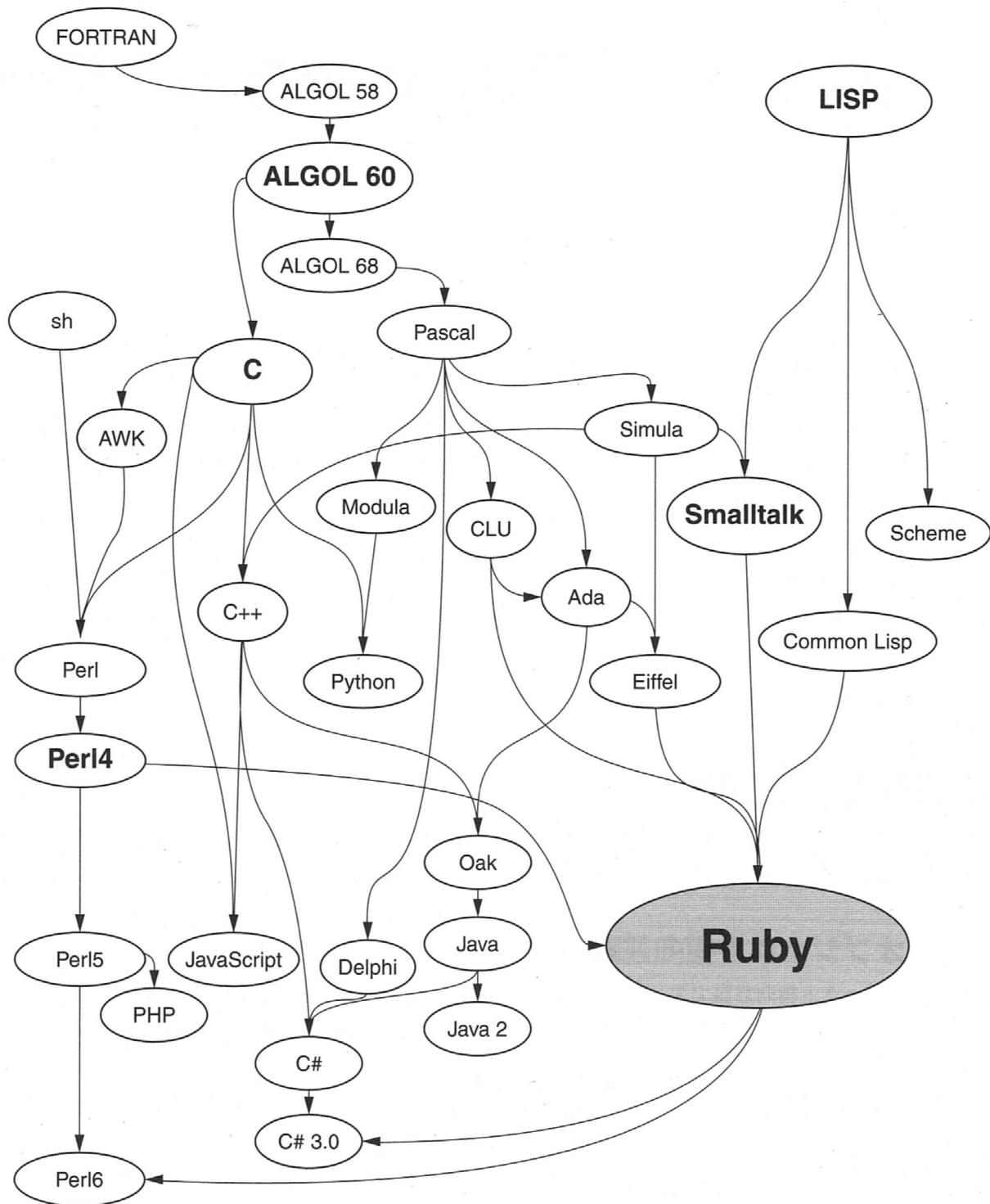
- ALGOL ヨーロッパ生まれのアルゴリズム記述言語。Pascal やC言語などに影響を与えたとされる。
- BCPL MULTICS で作成された高級言語。
- B言語初期の UNIX で作成されたインタプリタ方式の高級言語。BCPLを元に作られ、Cの原型となった。

## 継承・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。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。
- 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# Microsoft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。
- 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。
- Unified Parallel C 並列計算向けに C99 を拡張して作られた言語。

## 系統図

C 言語 と その流れを引く言語への系統図です。C が多くの言語へ影響を与えていることが分かります。



初めてのRuby, Yugui(著), オライリージャパン より引用

## データ型とは

コンピュータはメモリ上にある値を元に様々な計算を行います。メモリ上にはたくさんの 0 と 1 が存在します。この 0 と 1 の集まりをどのように解釈するか、それが「型」です。

C言語には、

- 文字を表すための型
- 整数を表すための型
- 実数（小数・浮動小数点数）を表すための型が用意されています。

## 変数の型

種類	型	説明	範囲
文字	char	1バイト文字（英数字など）を1字記憶できる	
文字列	char [] char *	文字列（複数の文字が集まつたもの）は文字型の配列として扱います	
整数	char	1バイトの符号付整数の値を記憶できる	-128～127 -2の7乗～2の7乗-1
	unsigned char	1バイトの符号なし整数の値を記憶できる	0～255 0～2の8乗
	short int	2バイトの符号付整数の値を記憶できる	-32768～32767 -2の15乗～2の15乗-1
	unsigned short int	2バイトの符号なし整数の値を記憶できる	0～65535 0～2の16乗-1
	int	4バイトの符号付整数の値を記憶できる	-21億4748万3648～21億4748万3647 -2の31乗～2の31乗-1
	unsigned int	4バイトの符号なし整数の値を記憶できる	0～42億9496万7296 0～2の32乗-1
	long	8バイトの符号付整数の値を記憶できる	-922京3372兆0368億5477万5808～922京3372兆0368億5477万5807 -2の63乗～2の63乗-1
実数	unsigned long	8バイトの符号なし整数の値を記憶できる	0～1844京6744兆0737億0955万1615 0～2の64乗-1
	float	4バイトの単精度浮動小数点実数（有効桁数 約7桁）	$\pm 1.175494 \times 10\text{の}-38\text{乗} \sim 3.402823 \times 10\text{の}38\text{乗}$
	double	8バイトの倍精度浮動小数点実数（有効桁数 約15桁）	$\pm 2.225074 \times 10\text{の}-308\text{乗} \sim 1.797693 \times 10\text{の}308\text{乗}$

【注】整数型で扱える範囲は環境（コンパイラや32bitパソコン／64bitパソコンなど）によって異なります。

## 定数の書き方

プログラム中で使用する定数の書き方を説明します。

種類	書き方 (例)	説明
文字	'a'	1バイト文字（英数字など）1文字
文字列	"abcd"	「"」で挟まれた文字の並び
整数(10進数)	156	10進数として扱われる
整数(8進数)	033	先頭が「0」の場合は8進数として扱われる
整数(16進数)	0x3f	先頭が「0x」の場合は16進数として扱われる
符号なし整数	358u	最後に「u」または「U」を付けると符号なし整数として扱われる
long型整数	1356L	最後に「L」を付けるとlong型整数として扱われる
float型実数	3.14f	最後に「f」または「F」を付けると单精度浮動小数点実数（有効桁数 約7桁）として扱われる。
double型実数	12.4567	倍精度浮動小数点実数（有効桁数 約15桁）として扱われる。

【注】たとえ1文字でも、'a'は「文字」、"a"は「文字列」として扱われる。

## 異なるデータ型での演算

### 暗黙の型変換

C言語で、異なる型のデータで演算をすると、精度の低い型のデータが精度の高い型に合わせられ、演算されます。

例えば

12 + 3.9

という演算の場合、12はint型、3.9はdouble型と見なされます。

演算（計算）の際には、int型のデータの12をdouble型のデータに変換して（12.0にして）、足されます。

結果は、double型で15.9となります。

char < short < int < float < double の順に精度が上がります。

被演算子	被演算子	精度の高い方の型	結果の型
char	int	int	int
short	long	long	long
int	float	float	float
int	double	double	double
float	double	double	double

## 明示的な型変換（キャスト）

精度の異なるデータ型同士の演算においては、暗黙の型変換が行われますが、プログラマが明示的に型変換を行うことも出来ます。

これを「キャスト」と言います。

(double)12 + 3.9

という演算の場合、12 は int 型ですが、プログラマが明示的に double 型に変換するよう指示しています。

結果は、double 型で 15.9 となります。

倉敷芸術科学大学 梶浦研究室(<http://www.kusa.ac.jp/~kajiura/c/data/newpage1.htm>)より  
引用・改変

# 書式指定子

## 出力用 フォーマット指定子一覧

フォーマット指定子とは、C言語のprintf()、fprintf()、sprintf()、scanf()、fscanf()、sscanf()などの関数で使用する、表示形式を指定するための記述子である。

指定子	対応する型	説明	使用例
%c	char	1文字を出力する	"%c"
%s	char *	文字列を出力する	"%8s", "%-10s"
%d	char, int, short	整数を10進数で出力する	"%-2d", "%03d"
%u	unsigned int, unsigned short	符号なし整数を10進数で出力する	"%2u", "%02u"
%o	int, short, unsigned int, unsigned short	整数を8進数で出力する	"%06o", "%03o"
%x	int, short, unsigned int, unsigned short	整数を16進数で出力する	"%04x"
%f	float, double	実数を出力する	"%5.2f"
%e	float, double	実数を指数表示で出力する	"%5.3e"
%g	float, double	実数を最適な形式で出力する	"%g"
%ld	long	倍精度整数を10進数で出力する	"%-10ld"
%lu	unsigned long	符号なし倍精度整数を10進数で出力する	"%10lu"
%lo	long, unsigned long	倍精度整数を8進数で出力する	"%12lo"
%lx	long, unsigned long	倍精度整数を16進数で出力する	"%08lx"

## 表示桁数の指定

表示桁数は<全体の幅>.<小数点以下の幅>で指定する。

どちらか片方だけの指定でも良いし、まったく指定しなくても良い。

指定がなければデフォルトが使用される。

<小数点以下の幅>は、文字列の場合には最大文字数の意味になる。

指定例	出力結果
printf("[%8.3f]", 123.45678);	[ 123.456]
printf("[%15s]", "I am a boy.");	[ I am a boy.]
printf(["%.6s"], "I am a boy.");	[I am a]
printf(["%8.3e"], 1234.5678);	[1.234e+3]

表示桁数の指定よりも実際の表示文字列が長くなることがある。これはsprintf()の場合に特に注意が必要である。

文字列の格納域と同じサイズにフォーマットを指定しても、配列オーバーを引き起こす可能性のあることを意味するからである。

## リーディングゼロの指定

数値フィールドの場合に、ゼロ詰めを指定することができる。桁数の指定のまえにゼロを付加する。

指定例	出力結果
printf(["%08.3f"], 123.45678);	[0123.456]
printf(["%05d"], 1);	[00001]

## 右詰・左詰の指定

デフォルトでは右詰となる。

左詰にしたいときは桁数指定の前にマイナスをつける。

指定例	出力結果
printf(["%-15s"], "I am a boy.");	[I am a boy. ]
printf(["%-8.3f"], 123.45678);	[123.456 ]
printf(["%-5d"], 1);	[1 ]

## 符号の指定

数値の表示は、デフォルトではプラス記号を出さない。

付けたいときは "+" を指定する。

指定例	出力結果
printf("[%+5d]", 32);	[ +32]
printf("[%+5d]", -32);	[ -32]
printf("[%+8.3f]", 1.414);	[ +1.414]

## 入力用 フォーマット指定子 一覧

scanf(), fscanf(), sscanf()などで使用する指定子である。

出力フォーマット指定子とほぼ同じだが、使えないものもある。

指定子	対応する型	説明
%c	char	1文字を入力する
%s	char *	文字列を入力する
%d	int	整数を10進数として入力する
%u	unsigned int	符号なし整数を10進数として入力する
%o	int, short, unsigned int	整数を8進数として入力する
%x	int, unsigned int	整数を16進数として入力する
%f	float	実数を入力する
%hd	short int	単精度整数を10進数として入力する
%ld	long	倍精度整数を10進数として入力する
%hu	unsigned short int	符号なし単精度整数を10進数として入力する
%lu	unsigned long	符号なし倍精度整数を10進数として入力する
%lo	long, unsigned long	倍精度整数を8進数として入力する
%lx	long, unsigned long	倍精度整数を16進数として入力する
%lf	double	倍精度実数を入力する

若葉プログラミング塾(<http://www.k-cube.co.jp/wakaba/server/format.html>)より引用・改変

## コラム

- コンピュータを創った人々
- コンピュータの仕組み
- 単位のお話
- 画素の話
- タッチタイピング
- ショートカットキー
- Unix コマンド

## コンピュータを創った人々

今日、世界で使われているコンピュータは、ノイマン式と呼ばれているものです。データとプログラムを同時にメモリ上におくという画期的なアイディアから生まれました。それまでは、それぞれの計算ごとに、配線をつなぎ変えていたのです！！ 大砲の弾道計算を行うときは、それよりの配線に、違う計算を行うときは、またそれようにと、何人もの人が何日もかけて、配線を組み直したものでした。それが、データとプログラムを同じコンピュータのメモリ上におくことで、ソフトを変えるだけで、同じ機械が何とおりもの機能を持つことを可能にしました！！ これってほんとにすごいことです。このアイディアから、今日のコンピュータ社会は始まりました。このアイディアを生み出した人が、ジョン・フォン・ノイマンです。どんな人だったのでしょうか？

### ジョン・フォン・ノイマン

(1903年12月28日 - 1957年2月8日) は、ハンガリー出身の数学者。ユダヤ系。現在のほとんどのコンピュータの動作原理であるストアードプログラム方式の考案者である。アラン・チューリング、クロード・シャノンらとともに、現在のコンピュータの基礎を築いた功績者で、現在使用されているコンピュータは「ノイマン型コンピュータ」と呼ばれる。

また、彼はゲーム理論におけるミニ・マックス法の発明者としてや、自己増殖オートマトンの考案、量子力学についての研究でも知られている。第二次世界大戦中にはマンハッタン計画に参加、爆縮レンズ開発に従事し、爆薬を32面体に配置することにより核爆弾が製造できることを10ヶ月に渡る計算により導いた。その圧倒的な計算力と、極めて広い活躍領域から、「悪魔の頭脳」と評された。

### クロード・シャノン

(1916年4月30日 - 2001年2月24日) はアメリカ合衆国の電気工学者、数学者。20世紀科学史における、最も影響を与えた科学者の一人である。情報理論の考案者であり、情報理論の父と呼ばれた。情報、通信、暗号、データ圧縮、符号化など今日の情報社会に必須の分野の先駆的研究を残した。アラン・チューリングやジョン・フォン・ノイマンらとともに今日のコンピュータ技術の基礎を作り上げた人物として、しばしば挙げられる。

1937年のマサチューセッツ工科大学での修士論文「継電器及び開閉回路の記号的解析」において、電気回路（ないし電子回路）が論理演算に対応することを示した。すなわち、スイッチのオン・オフを真理値に対応させると、スイッチの直列接続はANDに、並列接続はORに対応することを示し、論理演算がスイッチング回路で実行できることを示した。これは、デジタル回路・論理回路の概念の確立であり、それ以前の電話交換機などが職人の経験則によって設計されていたものを一掃し、数学的な理論に基づいて設計が行えるようになった。どんなに複雑な回路でも、理論に基づき扱えるということはコンピュータの実現に向けたとても大きなステップの一つだったと言える。

## アラン・チューリング

(1912年6月23日 - 1954年6月7日) はイギリスの数学者、論理学者、暗号解読者、コンピュータ科学者。チャーチ=チューリングのテーゼと計算可能性理論への貢献が、まず真っ先に挙げられる。特に、アルゴリズムを実行するマシンを形式的に記述したものの一つである「チューリングマシン」にその名を残し、人によっては前述のテーゼを「チューリング=チャーチ」と呼称するべきであるとする者もいるほどである。また、任意のチューリングマシンを模倣（エミュレート）できる「万能チューリングマシン」は、同分野の基本的な定理のひとつである停止性問題の決定不能性定理と関係する。さらに、理論面だけではなく、実際面でもコンピュータの誕生に重要な役割を果たした。コンピュータ科学および（チューリング・テストなどからは）人工知能の父とも言われる。

# コンピュータの仕組み

コンピュータって、どんな仕組みになっていて動くのでしょうか。今回は、コンピュータを形作る機械部分（ハード）とそれを動かす命令群（ソフト）、そして命令を作るプログラミング言語についての紹介です。

## ハード編

ハードとは、hardware。英語を直訳すると、「硬い物」 = 「パソコンを構成する様々な機器」のことです。

### CPU

Central (中央) Processing (処理) Unit (装置) の略。パソコンの頭脳で、様々な命令を処理します。キーボードから、どのキーが押されたか？画面のどこにどういう色を出すのか？など、いろいろな計算をしたり、絵を描いたり、みんなコンピュータにとっては命令です。

人に喩えると、「頭」の部分です。

CPUを作っている会社として、米国インテル社が有名です。CPUの速さの単位はGHz(ギガヘルツ)です。1GHzのCPUの場合、1秒間に10億回の足し算・引き算ができます。速ければ早いほどいいですが、最近のパソコンは性能が上がってきています。動画や画像処理、3Dゲーム等の用途であればともかく、ワープロやインターネットをするには、どれでも同じです。中古を購入する場合であれば、ここ2、3年以内のものがお薦めです。

### メモリ

メモリは、記憶 (memory) です。

CPUへの様々な命令を蓄えておいたり、CPUが命令を実行する際に必要となるデータを蓄えておく場所です。

人に喩えると、「作業机」です。単位は、GB(ギガバイト)。8 GBか16 GBあると快適です。

### ストレージ

SSD(ソリッドステートドライブ)やHDD(ハードディスクドライブ)の総称です。

いろいろなプログラムやデータを保存する場所です。人にたとえると、「倉庫」です。

SSDは記憶装置として半導体素子メモリを用いた補助記憶装置です。

HDDは磁性体を塗布した円盤を高速回転し、磁気ヘッドを移動することで、情報を記録し読み出す補助記憶装置です。

SSDは、HDDに比べると、物理的な稼働箇所がないため、読み出し書き込みの速度が速く、また省電力で静かです。反面HDDに比べると少し割高です。（Wikipediaより）

iPhone/iPadでは、SSDが使われています。単位はGB（ギガバイト）。64GBが主流です。

テレビを録画するためには、価格が安いことからハードディスクが用いられています。単位はTB（テラバイト）。1TBでおおよそ24時間録画したとして9日分の動画を保存できます。

## 光学ドライブ

CDやDVDドライブの総称です。インターネットの普及に伴い、ソフトや動画、音楽はダウンロードして愉しむものとなりましたが、昔のパソコンには標準で装備されていました。CDやDVDの読み取りや書き込みをする際に用います。

## マザーボード

以上の、CPU、メモリ、ストレージ、光学ドライブを取り付けるための基盤です。これにケースと電源を追加すれば、パソコンのできあがりです。

## ソフト編

「コンピュータ、ソフトなければ、ただの箱」といいますね。ソフト（software）は、ハードウェアに対して用います。

一般に使われているソフトという言葉の意味は、「コンピュータである機能を実現するための命令を集めたもの」という意味合いで使われています。パソコンソフトはこういった意味です。より広い意味では、パソコンを使いこなす知識やノウハウをも含めた意味で用います。

それでは、狭い意味でのソフトを扱っていきましょう。ソフトは、大別すると、OSとアプリケーションソフトに分けられます。

## OS (OperatingSystem)

OSとは、オペレーティング・システムの略です。パソコンを動かすために基本となる様々な機能を担当しているソフトで、そのため基本ソフトとも呼ばれます。

具体的には、

1. メモリやディスク、入出力や周辺機器などのハードウェアの管理を行う。
2. 利用者（ユーザー）がパソコンを操作するためのプログラム（ユーザーインターフェー

- ス) の提供する。
3. ファイルを開いたり、画面に表示したりなど、どのアプリケーションソフトにも共通する処理を、それぞれのアプリケーションソフトに提供する。
- などの機能を提供しています。

macOS や iOS, UNIX (ユニックス) /Linux (リナックス), Windows (ウィンドウズ) などがあります。

## アプリケーションソフト

アプリケーションソフトとは、コンピュータを使って特定の目的を果たすために作られたソフトウェアのことです。オペレーティングシステムなどの基本ソフトウェアに対して、応用ソフトウェアとも呼ばれます。

具体的には、ワープロソフト、表計算ソフト、ブラウザやメールソフト、お絵かきソフト、家計簿ソフト、ゲームソフトなどなどです。

プログラム開発に用いるエディタやコンパイラなども、アプリケーションソフトです。

## プログラミング言語

コンピュータはよく人に喻えられます。中国語では「電腦」といいます。コンピュータは、原理的には、人の論理的な思考のすべてが可能です。

コンピュータの特色は、

- 高速に計算できる！！（一秒間に数十億回も！！）
- 繰り返しが得意！！（1日でも1年でも飽きることなく同じことを繰り返します）
- 博覧強記！！（絶対忘れません！！）
- 命令どおりに動く（逆に言うと、気を回したり機転を利かすことはできないのですが、膨大な計算の結果、AI（人工知能）は人間らしく見えます）

コンピュータに、何かさせたいと思ったら、命令をする必要があります。例えば、

- $3 + 8$  を計算せよ
- 半径 3 cm の赤い円を描け
- 440 Hz(ラ) の音を出せ

などです。どうやったら命令できるのでしょうか？人間がコンピュータに命令を指示するために作られた言語が、プログラミング言語です。よく使われているところでは、Ruby（ルビー）、C（シー）、Java（ジャバ）などがあります。 Wikipedia に分かりやすく説明がありますのでお読み下さい。[プログラミング言語](#)

## Ruby

Ruby（ルビー）は、まつもとゆきひろ（通称 Matz）により開発されたオブジェクト指向スクリプト言語です。日本で開発されたプログラミング言語として初めて国際電気標準会議で国際規格に認証されました。開発者のまつもとゆきひろは、「Rubyの言語仕様策定において最も重視しているのはストレスなくプログラミングを楽しむことである (enjoy programming)」と述べています。(Wikipediaより)

Web界隈で広く使われており、フレームワーク *Ruby on Rails* も有名です。

たとえば、1から10までの合計の数を求めるには、

```
answer = 0
(1..10).each do |n|
    answer += n
end
print answer
```

と書くとanswer 55が求められます。

## C言語

略してC（しー）。FORTRANなどのプログラミング言語に比べて、効率がよく、より機械に近いところまで書き表すことができることから、主流となり、UNIXにおける標準開発環境です。

```
int n;
int answer = 0;
for(n = 1; n <= 10; n++){
    answer += n;
}
printf("%d\n", answer);
```

同じ、1から10までの合計を求めるプログラムですが、何となく、違いが感じられますか？

## Java

Java(ジャバ)は組み込みシステムや携帯機器（携帯電話・PHSやPDA・スマートフォン等）のシステムから、企業の情報システムを担う大規模なデータベース、サーバ、スーパーコンピュータまで、多くの分野で使用されている。(Wikipediaより)

```
int answer = 0;
for(int n = 1; n <= 10; n++){
    answer += n;
}
System.out.println("%d\n", answer);
```

## コンピュータの仕組み

### ノイマン式

プログラム内蔵方式のデジタルコンピュータで、CPUとアドレス付けされた記憶装置とそれらをつなぐバスを要素に構成され、命令（プログラム）とデータを記憶装置に記憶する方式。現在の主流となるコンピュータ・アーキテクチャである。(Wikipediaより)

### 量子コンピュータ

量子コンピュータ (quantum computer) は、量子力学的な重ね合わせを用いて並列性を実現するコンピュータ。従来のコンピュータ（以下「古典コンピュータ」）の基本素子は、情報量が0か1の何れかの値しか持ち得ない1ビットを扱うものであるのに対して、量子コンピュータでは量子ビット (qubit; quantum bit、キュービット) により、1キュービットにつき0と1の値を任意の割合で重ね合わせて保持する。 $n$  量子ビットあれば、 $2^n$  の  $n$  乗の状態を同時に計算できる。もし、数千qubitのハードウェアが実現した場合、この量子ビットを複数利用して、量子コンピュータは古典コンピュータでは実現し得ない規模の並列コンピューティングが実現する。理論上、現在の最速スーパーコンピュータで数千年かかる解けないような計算でも、例えば数十秒といった短い時間でこなすことができる、とされている。(Wikipediaより)

ITエンジニアのための量子コンピュータ入門(<http://codezine.jp/article/corner/629>)なども参考にして下さい。

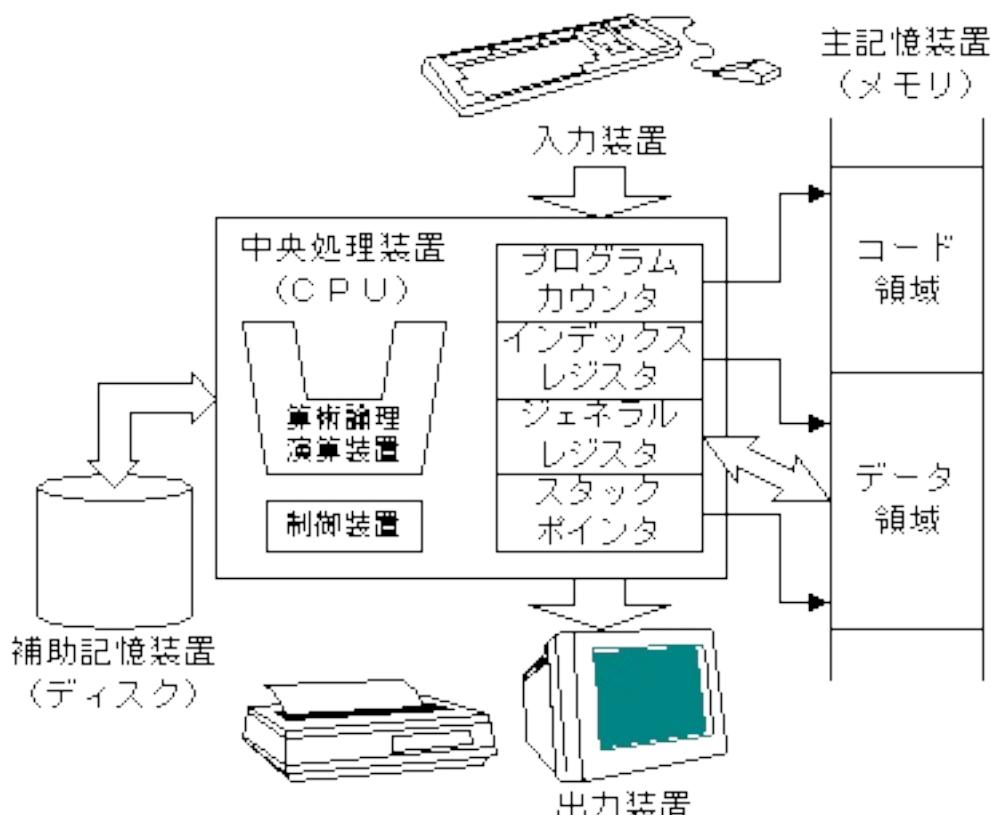
### ニューラルネットワーク

ニューラルネットワーク(神経回路網、英: neural network, NN)は、脳機能に見られるいくつかの特性を計算機上のシミュレーションによって表現することを目指した数学モデルである。研究の源流は生体の脳のモデル化であるが、神経科学の知見の改定などにより次第に脳モデルとは乖離が著しくなり、生物学や神経科学との区別のため、人工ニューラルネットワーク(人工神経回路網、artificial neural network, ANN)とも呼ばれる。(Wikipediaより)

人工知能（A I）に興味がある方であれば、ディープラーニング（深層学習）という言葉を聞かれたこともあるかと思います。学習したネットワークをもとに創られたCPUもあります。

## コンピュータ(computer)の5大機能と動作原理

機能	働き	装置	備考	
演算機能	考える	演算装置	中央処理装置 (CPU) Central Processing Unit	cpu
制御機能	動かす	制御装置		
記憶機能	覚える	記憶装置 memory device	主記憶装置	メモリ
			補助記憶装置	ストレージ (SSD, HDD など)
入力機能	取り込む	入力装置	入出力装置 (I/O) Input/ Output device	キーボード, マウス など
出力機能	外に出す	出力装置		ディスプレイ, プリンタ など



- プログラム (コンピュータを動かす一連の命令) (program) はデータ (data) と共にメモリ (memory) に格納されている
  1. 命令フェッチ (instruction fetch) : メモリ中のプログラムを読み出し
  2. 命令デコード (instruction decode) : 命令を解釈 (復号化) し
  3. エグゼキューション (execution) : 実行する

を繰り返す

- 次に実行する命令が格納されている場所を指すレジスタ（register）をプログラムカウンタ（program counter）という
- サブルーチン（subroutine）からの戻り番地を保持するためにスタック（stack）を用いる
- CPUが解釈実行できるプログラムを機械語（machine language）プログラムという
- 次のような命令がある
  - データをメモリからレジスタへロード（load）
  - データをレジスタからメモリへストア（store）
  - レジスタ間で算術論理演算
  - ステータスレジスタの内容に従って条件分岐
  - サブルーチンコール（subroutine call）、リターン（return）

## プログラミング言語（programming language）について

### アセンブリ言語（assembly language）

低水準言語（低級言語）（low-level language）とも呼ばれる。  
（※低級とは、ハードウェアに近いという意味）

- 機械語は2進数で表現（符号化）されているので人間が記述するのは難しい。  
そこで、その意味を表現するニーモニックコード（mnemonic code）で記述するのがアセンブリ言語。
- 機械語とほぼ1対1に対応する
  - LOAD R0, 1234
  - STORE 2345, R0
  - SUB R0, R1
  - JUMP NE, 3456
  - CALL 4567
- アセンブリ言語で書かれたプログラムを機械語に変換するプログラムをアセンブラー（assembler）と呼ぶ

### 高水準言語（高級言語）（high-level language）

- アセンブリ言語（機械語）はCPUに依存する上、部品としての粒度が小さいため、可読性、移植性が低い。  
そこで、人間にとて分かりやすい命令と構文規則からなる言語が用いられる。
  - BASIC
  - FORTRAN
  - COBOL
  - Pascal
  - C
  - Lisp
  - Prolog

- Smalltalk
  - Java
  - Ruby
- 等
- 高水準言語で記述されたプログラムを機械語に変換するトランスレータ (translator) としては、コンパイラ (compiler) やインタプリタ (interpreter) が用いられる。
  - コンパイラは翻訳に、インタプリタは通訳に似ている。
  - これらは言語処理系あるいは単に処理系と呼ばれる。

## プログラミング言語C

UnixというOS (Operating System) を開発するために作られた言語

### C

Kenneth L. Thompson と Dennis M. Ritchie が B 言語を改良して作った。 (1973)

### K&R C

brian w. kernighan & dennis m. ritchie の 「the c programming language」 (1978)

### C99, C11

ISOで規格を改訂し、c++の機能のいくつかを取り込んだ。 (1999, 2011)

神戸女学院大学 出口 弘(<http://wwws.kobe-c.ac.jp/deguchi/c/>より引用・改変)

## プログラム開発の手順

さまざまな開発手法があるが、ここでは、ウォーターフォールモデルの例を挙げる。

1. プログラムの仕様決定（外部設計）
  - i. 課題を明確にする
  - ii. 解決のためのアイデアを練る
  - iii. 必要な仕様を決定する
2. プログラム設計（内部設計）
  - i. データ構造、アルゴリズムを設計する
  - ii. フローチャート等を作成する
3. プログラムのコーディング
  - i. プログラミング言語の文法に従って、プログラムを書く
4. プログラムのデバッグ・テスト
  - i. 文法エラー、論理エラーを取り除く。
  - ii. 単体テスト、結合テストを行う。
  - タイプミス等の修正を行う。
  - 「外部設計」で顧客の要求仕様を十分に検討し、「内部設計」で処理手順を十分に検討しているとよいが、そうでなかった場合、この段階において出戻りが発生する場合がある。
5. プログラムのドキュメント作成
  - 各工程でドキュメント作成を並行して進めるとよい。
  - ドキュメントには以下の2種類ある

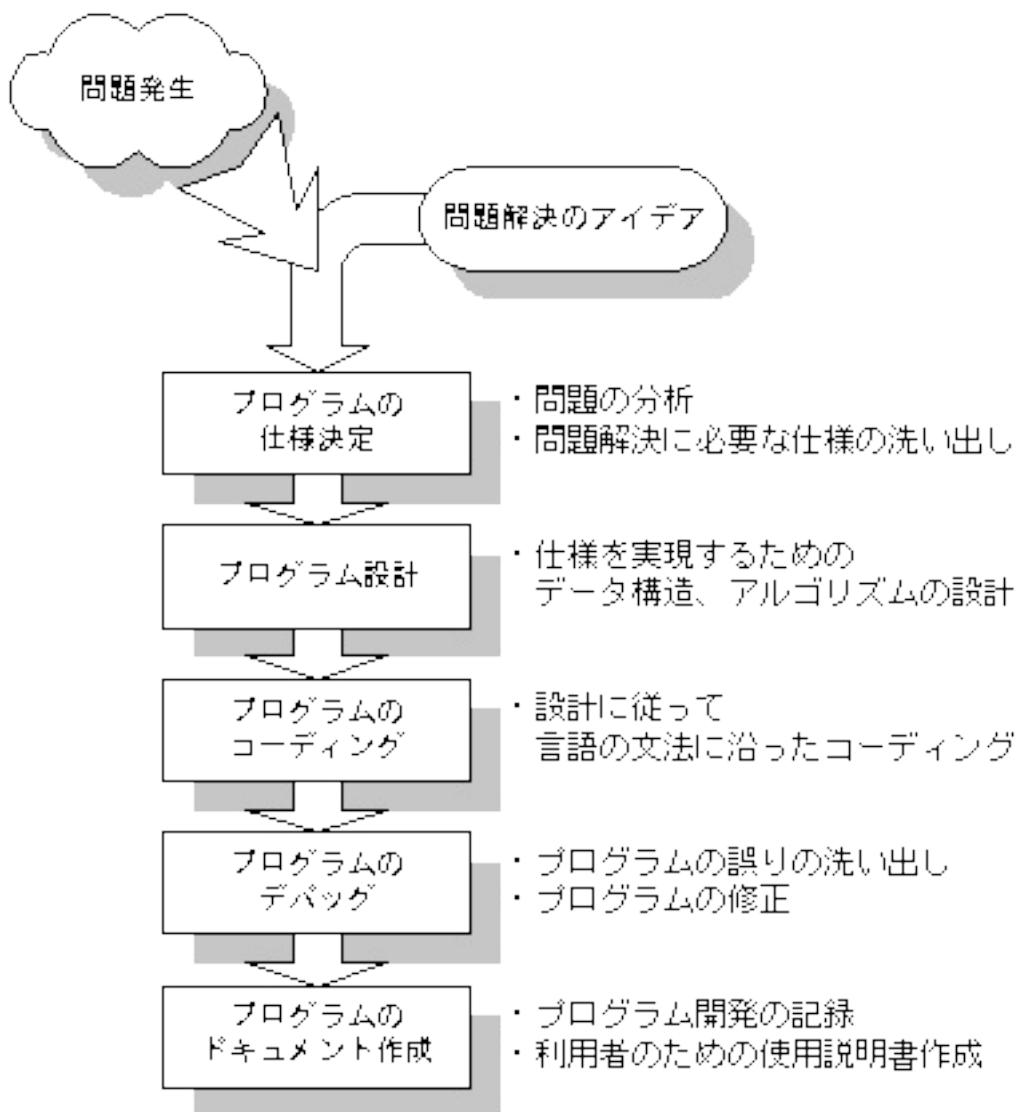
### システムレベルドキュメント

プログラムの保守、管理のための技術的情報の記録

(ソースコードからドキュメントを自動生成する手法もある)

### ユーザレベルドキュメント

プログラム利用者のための使用説明書



## アルゴリズムとデータ構造

プログラム = アルゴリズム + データ構造

### アルゴリズム (*algorithms*)

簡単には、問題を解決する手順や計算方法のことであり、 算法や解法と訳されることもある。

### データ構造 (*data structures*)

計算機内部にデータを蓄える形式のこと。

詳しくは後述

## コンパイル (compile) & リンク (link) とファイル (file)

- c 言語のソースプログラム (source program) は純粋なテキストファイルで、その拡張子は .c である。ソースファイル (source file) とも呼ばれる。
- コンパイラでコンパイルしてオブジェクトファイル (object file) を得る。

- ・ リンカでライブラリ (library) とリンクして実行ファイル (executable file) を得る。
- ・ ライブラリとして標準ライブラリ関数等が予め用意されている。

## ファイル名

	Unix(Mac/Linux)の場合	Windowsの場合
ソースプログラム	ファイル名 <b>.c</b> 例： program.c	ファイル名 <b>.c</b> 例： program.c
オブジェクトプログラム	ファイル名 <b>.o</b> 例： program.o	ファイル名 <b>.obj</b> 例： program.obj
実行プログラム	ファイル名 <b>.out</b> 例： program.out	ファイル名 <b>.exe</b> 例： program.exe

## コンパイルコマンド

gcc	gcc -o ファイル名 ファイル名.c 例： gcc -o program program.c
make	make ファイル名 例： make program (大規模で複数のプログラムで構成される際に用いる)

## 実際のプログラミングの流れ

1. ソースプログラムはテキストエディタ(Atomなど)で作成、修正する
2. コンパイルしてオブジェクトファイルを得る  
構文チェックによって、検出された文法エラーは修正する
3. ライブラリとリンクして実行ファイルを得る（自動でリンクされることが多い）  
検出された未定義オブジェクト等は修正する
4. プログラムを実行し、テストする
5. エラーがあれば修正する

神戸女学院大学 出口 弘(<http://wwws.kobe-c.ac.jp/deguchi/c/>より引用・改変)

# アルゴリズムとデータ構造

プログラム = アルゴリズム + データ構造

## アルゴリズム

アルゴリズム（英: algorithm）とは、数学、コンピューティング、言語学、あるいは関連する分野において、問題を解くための手順を定式化した形で表現したものを使う。「算法」と訳されることもある。

「問題」はその「解」を持っているが、アルゴリズムは正しくその解を得るために具体的な手順および根拠を与える。さらに多くの場合において効率性が重要となる。（Wikipediaより）

## データ構造

データ構造（データこうぞう、英: data structure）は、計算機科学において、データの集まりをコンピュータの中で効果的に扱うため、一定の形式に系統立てて格納するときの形式のことである。

ソフトウェア開発において、データ構造についてどのような設計を行うかは、プログラム（アルゴリズム）の効率に大きく影響する。そのため、さまざまなデータ構造が考え出されている。

## 基本的なデータ構造

- 配列
- スタック
- キュー
- 連想配列
- ハッシュテーブル / ルックアップテーブル
- 線形リスト
- 木構造
- グラフ

（Wikipediaより）

## 単位のお話

パソコン買おうと思っているんだけれど、CPUが3GHz（ギガヘルツ）で、メモリが4GB（ギガバイト）で、ハードディスクが1TB（テラバイト）で・・・。よく耳にしますよね。どういった意味なのでしょう。

ビット(bit)、バイト(Byte)、キロバイト(kB)、メガバイト(MB)、ギガバイト(GB)、テラバイト(TB)

パソコンでよく使われる単位のお話です。

- 1ビット(bit)

コンピュータで扱うことができる最小の情報量です。スイッチが入っている (=1) か、切れている (=0) かなど、2通りの状態を表すことができます。2進法の1桁分の情報量です。

10進法では、10個あつまると、次の位へ繰り上がりますが、2進法では、2つ集まるところの位へ繰り上がります。ですから、10進法の1桁では、0～9までの10通りの状態を表すことができましたが、2進法の1桁では、0～1までの2通りの状態を表すことができます。

2桁分 (2ビット) では、00, 01, 10, 11 の4通りを表すことができます。

3桁分 (3ビット) では、000, 001, 010, 011, 100, 101, 110, 111 の8通りを表すことができます。

- 1バイト(Byte)

半角文字1文字分 (8ビット = 2進法の8桁分) の情報量です。

つまり、00000000～11111111までの256通りの情報量です。256通りありますから、数字(0 - 9) や、アルファベット(A - Z, a - z)、記号(@, #, !など)を区別するには十分です。

例えば 半角の大文字のアルファベット "A" に "01000001" という2進数を、"B" には "01000010" を対応させて、アルファベットを表します。

- 2バイト全角文字1文字分の情報量です。1バイトでは 256通りの文字を区別することができますが、「ひらがな」や「カタカナ」、「漢字」などから成り立つ日本語を表記するには、不十分です。

そこで2バイトにすることにより、 $256 \times 256 = 65536$ 通りの文字を区別することができるようになります。

従来は、Shift-JIS(シフトジス)と呼ばれる文字体系で、日本語を表現していましたが、近年では、世界中の文字を3バイトで表現するUTF-8(ユーティーエフエイト)が主流となってきています。

- 1キロバイト(kB) = 1,024バイト

2進数10桁(=2の10乗)で1024通りを表現できます。

10進法で、1,000倍のことを k (キロ) といいます。1000 g は 1kg ですね。コンピュータの世界では、1024でひとまとめにして、1024 バイト のことを 1kB といいます。

おおよそ原稿用紙1枚、A4用紙一枚程度の情報量です。

- 1メガバイト(MB) = 1,024 キロバイト = 1,048,576 バイト

小さめの写真約1枚分、A4約1000枚分の情報量です。昔懐かしいフロッピーディスク1枚は1.44MBでした。

- 1ギガバイト(GB) = 1,024 メガバイト = 1,048,576 キロバイト = 1,073,741,824 バイト

CD 1枚74分で、640MB (≈0.64GB) 、DVD 1枚2時間で、4.7GB です。

- 1テラバイト(TB) = 1,024 ギガバイト = 1,048,576 メガバイト = 1,073,741,824 キロバイト = 1,099,511,627,776 バイト

最近のハードディスクは大容量です。大量の動画を撮りためて、毎日2時間ずつ動画を見るなら 108 日分 (3ヶ月半) 分の容量になります。

## おまけ

- 国際単位系(SI単位系)での接頭辞一覧

1,000 m = 1 km

100 a = 1 ha

1 dL = 1 / 10 L

1 cm = 1 / 100 m

1 mm = 1 / 1,000 m

などがお馴染みですね。

ほかにはどんな記号があるのでしょう？

接頭辞	接頭辞	記号	乗数
ヨタ	yota	Y	10の24乗=1,000,000,000,000,000,000,000,000倍
ゼタ	zeta	Z	10の21乗=1,000,000,000,000,000,000,000倍
エクサ	exa	E	10の18乗=1,000,000,000,000,000,000倍
ペタ	peta	P	10の15乗=1,000,000,000,000,000倍
テラ	tera	T	10の12乗=1,000,000,000,000倍
ギガ	giga	G	10の9乗=1,000,000,000倍
メガ	mega	M	10の6乗=1,000,000倍
キロ	kilo	k	10の3乗=1,000倍
ヘクト	hecto	h	10の2乗=100倍
デカ	deca	da	10の1乗=10倍
			10の0乗=1倍
デシ	deci	d	10の-1乗=10分の1
センチ	centi	c	10の-2乗=100分の1
ミリ	milli	m	10の-3乗=1,000分の1
マイクロ	micro	μ	10の-6乗=1,000,000分の1
ナノ	nano	n	10の-9乗=1,000,000,000分の1
ピコ	pico	p	10の-12乗=1,000,000,000,000分の1
フェムト	femto	f	10の-15乗=1,000,000,000,000,000分の1
アト	atto	a	10の-18乗=1,000,000,000,000,000,000分の1
ゼプト	zepto	z	10の-21乗=1,000,000,000,000,000,000,000分の1
ヨクト	yocto	y	10の-24乗=1,000,000,000,000,000,000,000,000分の1

# 画素の話

## ドロー系とペイント系

お絵かきソフトには、大きく分けて2種類あります。

- ドロー系 Affinity Designer, Illustrator etc
- ペイント系 Affinity Photo, Photoshop etc

点と線、つまりベクトルで描かれる画像データをベクタデータと呼び、そうしたデータで絵を描くソフトをドローソフトと呼びます。拡大しても画像が劣化しないという特徴があります。

これに対し、ドットで絵を描くツールをペイントソフトと呼びます。(ベクタデータに対してラスタデータといいます。) こちらは拡大するとジャギー(ギザギザ)が出てしまいます。(ASCIIデジタル用語辞典より引用・改変)

## ベクタデータとラスタデータ

ドロー系ソフトで、赤い丸を書いた場合は、データはベクタデータとして保持されます。つまり、画面の左から5cm、上から10cmの点を中心とした半径3cmの円、色は赤、といった具合に、ベクタデータとして保存されます。ですから、この丸を2倍に拡大しても、「半径6cmの円を描く」ことになるだけですので、ギザギザになることはありません。

一方、ペイント系ソフトで、赤い丸を書いた場合には、データはラスタデータとして保存されます。ちょうど、1mmづつの方眼紙が縦に100個、横に100個集まって、一枚の絵が構成されていると想像してください。そして、左上から1番目、上から1番目の方眼は赤色、左上から1番目、上から2番目の方眼は黄色、左上から1番目、上から3番目の方眼は黄色、・・・左上から100番目、上から100番目の方眼は青色。という具合です。

ですから、絵を2倍に拡大すると、今まで1mmだった方眼が2mmの方眼に変わることですから、ギザギザが目立つことになります。

## 画素数、ppiについて

画素とは、画像を構成する最小単位のことです。ピクセルとも言います。先ほどの例では、1mmの方眼が画素でしたが、実際の方眼のサイズは1mmではありません。1インチ=2.54cmの間にいくつ画素があるかを、ppi(pixel per inch)と言います。

iPhone 6s Plus ですと、縦6.8インチ(122mm)、横2.7インチ(68mm)の間に、1920\*1080ピクセルありますから、1インチの間には、401個 画素があることになります。401ppiって目に見えないです。すごいですね。

iPhone 6s / 6s Plus には1200万画素(4032\*3024)のデジカメが搭載されています。といったときには、一枚の写真が、1200万個の画素で構成されていることを意味しています。

つまり、横4032画素×横3024画素=1200万画素に一枚の写真を分割して、

左から1番目、上から1番の方眼は赤色。  
左から1番目、上から2番の方眼は黄色。  
左から1番目、上から3番の方眼は黄色。  
...  
左から4032番目、上から3024番の方眼は青色。

というようにして写真が保存されるということです。

## 写真のファイルサイズ

1200万画素のデジカメは、横4032×縦3024=1200万画素でした。そして、それぞれの画素には、赤、白、黄色など単純な色ばかりではなく、桜色や若草色など微妙な色もあります。光の三原色は、赤と緑と青です。それぞれの光の強さを0～255までの256段階で表現することにすると、16777216色(=256×256×256)もの微妙な色合いを表現することが出来ます。

そして、この写真を保存するために必要なファイルの大きさは、 $4032 \times 3024 \times 3$  バイト = 36,578,304 バイト ≈ 34.9メガバイトとなります。

## 色の表し方について

色の三原色は、赤、青、黄ですが、コンピュータでは、光の三原色、Red(赤)、Green(緑)、Blue(青)を用いて絵を表します。(RGBモデルといいます)

他に、色相(Hue)、彩度(Saturation)、明度(Brightness)によって色を表現するHSBカラーモデルなどもあります。色彩画像関係にご興味のある方は調べてみて下さい。

Webサイトで用いられるhtmlでは色を表すのに、#ffffff(白色)といった表記をします。この表記は何でしょうか？

今、舞台を照らす3つのスポットライト、Red(赤)、Green(緑)、Blue(青)があるとします。

Red(赤)が消えていて、Green(緑)が消えていて、Blue(青)が消えているとき、舞台は黒色になります。(何も光がないのですから当然ですね)

Red(赤)が消えていて、Green(緑)が消えていて、Blue(青)がついているとき、舞台は青色になります。(青がついているのですから当然ですね)

Red(赤)が消えていて、Green(緑)がついていて、Blue(青)が消えているとき、舞台は緑色になります。(緑がついているのですから当然ですね)

Red(赤)が消えていて、Green(緑)がついていて、Blue(青)がついているとき、舞台は水色になります。

Red(赤)がついていて、Green(緑)が消えていて、Blue(青)が消えているとき、舞台は赤色になります。

Red(赤)がついていて、Green(緑)が消えていて、Blue(青)がついているとき、舞台は紫色になります。

Red(赤)がついていて、Green(緑)がついていて、Blue(青)が消えているとき、舞台は黄色になります。

Red(赤)がついていて、Green(緑)がついていて、Blue(青)がついているとき、舞台は白色になります。

文章で書くと複雑ですが、それぞれのランプがついていることを、1ランプが消えていることを、0で表すことにすると、それぞれのランプの状態(ついているか消えているか)と、舞台の色との関係は次の表で表されます。

R(赤)	G(緑)	B(青)	色	二進数
0	0	0	黒	0
0	0	1	青	1
0	1	0	緑	2
0	1	1	水色	3
1	0	0	赤	4
1	0	1	紫	5
1	1	0	黄	6
1	1	1	白	7

ここでは、消えている、ついている、という2つの状態を表すだけでしたので、0と1 2進法1桁で十分でしたが、実際には、とっても明るくついている時もあるでしょうし、ほとんど消えそななくらいの明るさでついているときもあります。

そこで、明るさが変わるように、それぞれの電球は、2V(ボルト)の電池と、1V(ボルト)の電池につながっているとしましょう。

すると、2Vの電池につながっていなくて、1Vの電池にもつながっていないときには、電球の明るさは最低です。(電球がつかないので真っ暗です)

2Vの電池につながっていなくて、1Vの電池につながっているときには、電球の明るさはちょっと光ります。

2Vの電池につながっていて、1Vの電池にはつながっていないときには、電球の明るさはかなり光ります。

2Vの電池につながっていて、1Vの電池にもつながっているときには、電球の明るさは最高です。

先ほどと同じように、電池のつなぎ具合を表にすると、

2Vの電池	1Vの電池	電球の明るさ	電球にかけられている電圧
0	0	レベル0 (最低)	$2V \times 0 + 1V \times 0 = 0V$
0	1	レベル1 (ちょっと明るい)	$2V \times 0 + 1V \times 1 = 1V$
1	0	レベル2 (かなり明るい)	$2V \times 1 + 1V \times 0 = 2V$
1	1	レベル3 (最高に明るい)	$2V \times 1 + 1V \times 1 = 3V$

このように、2Vの電池と、1Vの電池を使うと、0V～3Vまでの4通りの明るさを表現することができました。

同様に、4Vの電池と、2Vの電池と、1Vの電池を使うと、0V～7Vまでの8通りの明るさを表現することができます。

同様に、128Vの電池と、64Vの電池と、32Vの電池と、16Vの電池と、8Vの電池と、4Vの電池と、2Vの電池と、1Vの電池（合計8種類の電池）を使うと、0V～255Vまでの256通りの明るさを表現することができます。

128V	64V	32V	16V	8V	4V	2V	1V	電圧
0	0	0	0	0	0	0	0	0V
0	0	0	0	0	0	0	1	1V
0	0	0	0	0	0	1	0	2V
								(中略)
1	1	1	1	1	1	1	0	254V
1	1	1	1	1	1	1	1	255V

このようにそれぞれの桁の重みが2倍になっていくのが2進法です。

（10進法ではそれぞれの桁の重みは10倍になります。1円玉より10円玉は10倍の重みがあります。100円玉は10円玉より10倍の重みがあります。・・・）

この2進法で、1桁のことを1ビット(bit)、8桁のことを1バイト(Byte)といいます。つまり1ビットでは、0と1の2通りの状態を表現できますし、1バイトでは、0から255までの256通りの状態を表現できます。

もとのRed(赤),Green(緑),Blue(青)のランプの話に戻すと、それぞれのランプが全くついていない(0)～最高に明るく輝いている(255)までの段階を表すために1バイトずつ、計3バイト必要となります。

そして、赤、緑、青 それぞれで、256段階光の強さがありますから、全体では、3バイト  
 $256 \times 256 \times 256 = 16777216$ 色を表現することが出来ます。

16進数2桁で表現すると、#fffff は、Rがff, Gがff, Bがffで、全部光っている状態のことですね。つまり、白です。

## クイズ

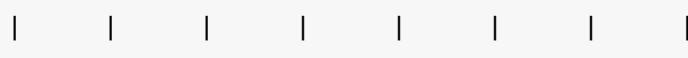
7日分の給料の支払いとして金の延べ棒が1本ある。

日払い希望のため、1日分ごとに線を引いた。

6回はさみを入れて切り取れば、金の延べ棒は7等分されるので日払い可能である。

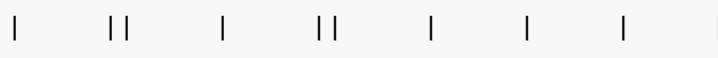
しかし、金の延べ棒を6回も切り取るのは大変なため、切り取り回数は2回にしたい。  
どことどこを切り取ればよいか。

(金の延べ棒)



(答) 1と2と4に分割すれば良いですね。

(金の延べ棒)



421

001 -> 1

010 -> 2

011 -> 3

100 -> 4

101 -> 5

110 -> 6

111 -> 7

## タッチタイピングのおすすめ

タイピングができるようになると、コーディングがスムーズに行えるようになります。書きたいことを書くためにキーの場所を探しているようだと、思考の流れが止まってしまいます。

また、プログラミング以外にも文章を作成したりすることも速くなります。何よりパソコンを使うのが楽しくなります。

タッチタイプは、一日30分×1週間でマスターできます。練習のポイントは、それぞれの指のホームポジション（初期位置）においてた指が、ほかのキーをタイプしたときにも、それぞれの指のホームポジションに戻ってくることです。

インターネットでタイピング練習(<http://www.e-typing.ne.jp>)のようなサイトもありますので、一分間に100文字くらいはタイプ出来るようにスキルを身につけましょう。

## ショートカットキー

マウスとキーボードのいったりきたり、鬱陶しくありませんか？  
キーボードから手を離さなくても可能になるショートカットキーのご紹介です。  
ショートカットキーをマスターして、生産性を向上させましょう。

### 一般的なショートカット

目的	キー操作 (Mac)	キー操作 (Win)	Memo
選択した項目をコピーする	Cmd + C	Ctrl + C	CopyのC
選択した項目を切り取る	Cmd + X	Ctrl + X	鉄の形
選択した項目を貼り付ける	Cmd + V	Ctrl + V	挿入しているイメージ
元に戻す	Cmd + Z	Ctrl + Z	
ファイルを新規作成する	Cmd + N	Ctrl + N	NewのC
ファイルを開く	Cmd + O	Ctrl + O	OpenのO
ファイルを保存する	Cmd + S	Ctrl + S	SaveのS

### atom のショートカット

プログラマであれば、エディタのショートカットキーを熟知しましょう。(by 達人プログラマ)

### 基本

操作	キー	備考
コマンドパレット	cmd-shift-P	Pallet
設定画面	cmd-,	
タブを閉じる	cmd-W	
スニペット表示	alt-shift-S	Snippet

## 保存・開く

操作	キー	備考
上書き保存	cmd-S	Save
別名で保存	cmd-shift-S	
開いているファイルを全て保存	cmd-alt-S	
プロジェクトを追加	cmd-shift-O	Open

## ファイルを開く

操作	キー	備考
プロジェクトからファイルを探す	cmd-T or cmd-P	
現在開いてるファイルから探す	cmd-B	
コミットされていないgitファイルから探す	cmd-shift-B	

advanced-open-file パッケージを導入すると cmd-alt-o で開ける

## ツリーView

操作	キー	備考
ツリーViewの表示・非表示	cmd- \	
ツリーViewにフォーカス	ctrl-0	
(ツリーView上で) 追加	A	Add
(ツリーView上で) 名前変更	M	Modify
(ツリーView上で) 削除	Delete	

## ファイル設定

操作	キー	備考
エンコードの種類を変更	ctrl-shift-U	
ファイルの種類（シンタックス）を変更	ctrl-shift-L	

## 移動

操作	キー	備考
単語単位の移動	alt-B / alt-F	Back / Forward
行の先頭・末尾	ctrl-A / ctrl-E	? / End
ファイルの先頭・末尾	cmd-up / cmd-down	

## ジャンプ

操作	キー	備考
行・桁番号でジャンプ	ctrl-G ( 行:桁 で移動)	Go
シンボルで移動	cmd-R	

## 選択

操作	キー	備考
全選択	cmd-A	All
1行選択	cmd-L	Line
現在の単語を選択	ctrl-shift-W	Word

## 編集

操作	キー	備考
カーソル前後の文字を入れ替え	ctrl-T	
次の行と結合	cmd-J	Join
行の移動	cmd-ctrl-up / cmd-ctrl-down	
行の複製	cmd-shift-D	Duplicate
現在の単語を大文字に	cmd-K → U	Upper
現在の単語を小文字に	cmd-K → L	Lower

## 削除・カット

## ショートカットキー

操作	キー	備考
1行削除	ctrl-shift-K	
末尾までカット	ctrl-K	
単語の先頭・末尾まで削除	alt-H / alt-D	

## マルチカーソル

操作	キー	備考
カーソル追加	cmd-click	
複数行の選択からマルチカーソルへ	cmd-shift-L	
次の同じ単語を追加選択	cmd-D	
同じ単語を全選択	ctrl-cmd-G	

## ブラケット（括弧）

操作	キー	備考
対応する括弧へ移動	ctrl-M	
現在のブロックを選択	ctrl-cmd-M	

## 検索

操作	キー	備考
現在のタブから検索	cmd-F	Find
プロジェクトから検索	cmd-shift-F	Find
(検索したあとで) 前・次の検索結果へ	cmd-shift-G / cmd-G	

## 折りたたみ

操作	キー	備考
折りたたむ	alt-cmd-[	
開く	alt-cmd-]	
全て折りたたむ	alt-cmd-shift-[	
全て開く	alt-cmd-shift-]	

## パネル

操作	キー	備考
パネル分割	cmd-K → カーソルキー	
パネル間の移動	cmd-K → cmd-カーソルキー	

## Markdown

操作	キー	備考
Markdownプレビュー表示	ctrl-shift-M	

Markdownは以下のスニペットが用意されている

- img
- table
- b (bold)
- i (italic)

キーボードショートカット個人的まとめ (<http://qiita.com/YusukeHosonuma/>) より引用・  
改変

## Unix コマンド

Windows 10 にも bash と呼ばれるシェルが搭載されるようになりました。  
技術者として基本的なUNIXコマンドは習得しておきたいものです。  
以下のサイトに紹介がありますので、是非習得されて下さい。

これだけは覚えておきたい！基本的なUNIXコマンド20【初心者向け】 (<http://techacademy.jp/magazine/6406>)

## 学習に役立つサイトのご紹介

いろいろ役立つサイトのご紹介です。

### wikipedia

言わずと知れた、インターネット上の百科事典です。コンピュータに関する用語も豊富に記載されており、学習に役立ちます。

### Qiita

プログラミング知識を共有しよう。

### Codezine

開発者のための実装系Webマガジン

### ITpro

日経BP社が運営する、IT（情報技術）にかかるプロフェッショナルに向けた総合情報サイト。ニュースだけでなく、詳細な解説／コラムやネットの双方向性を活用したコンテンツを提供。

### git

git（ギット）は、プログラムのソースコードなどの変更履歴を記録・追跡するための分散型バージョン管理システムである。Linuxカーネルのソースコード管理に用いるためにリナス・トーバルズによって開発され、それ以降ほかの多くのプロジェクトで採用されている。Linuxカーネルのような巨大プロジェクトにも対応できるように、動作速度に重点が置かれている。  
(Wikipediaより)

#### サルでもわかるGit入門

また、ソフトウェア開発プロジェクトのための共有ウェブサービスとして、

- [GitHub](#)
- [Bitbucket](#) が有名です。

## C言語入門(全22回) - プログラミングならドットインストール

### 正規表現

正規表現（せいきひょうげん、英: regular expression）とは、文字列の集合を一つの文字列で表現する方法の一つである。もともと正規表現は形式言語理論において正規言語を表すための手段として導入された。その後正規表現はテキストエディタ、ワードプロセッサをはじめとするアプリケーションでパターンマッチ文字列を表すために使用されるようになり、表せるパターンの種類を増やすために本来の正規表現にはないさまざまな記法が新たに付け加えられた。アプリケーションやプログラミングにおいて正規表現を用いた文字列のパターンマッチを行う機能のことを、単に正規表現という。（Wikipediaより）

正規表現の練習に最適 [Rubular](#)

### Markdown

Markdown は、文書を記述するための軽量マークアップ言語のひとつである。「書きやすくて読みやすいプレーンテキストとして記述した文書を、妥当なXHTML(もしくはHTML)文書へと変換できるフォーマット」として、ジョン・グルーバー(John Gruber)とアーロン・スワース(Aaron Swartz)によって考案された。Markdownの記法の多くは、電子メールにおいてプレーンテキストを装飾する際の慣習から着想を得ている。

[文章作成やメモ書きにも便利、Markdown記法](#)

### SQLite

様々なデータベースがあります。

ここでは、初心者向けに学習しやすいSQLiteをご紹介します。

[SQLite入門 \(<http://www.dbonline.jp/sqlite/>\)](#) で学習されて下さい。

### 英語翻訳

プログラマとして、簡単な英語は知っておきたいものです。翻訳の実例もついていますので、学習に役立つかと思います。

[英語翻訳 \(<http://translate.weblio.jp>\)](#)

### ATOK

30年以上の歴史ある日本語入力ソフトです。高い変換精度、言葉遣いのチェック、英語への変換機能などあり、月額300円～利用出来ます。[\(http://atok.com/indexb.html\)](http://atok.com/indexb.html)

## 学習に役立つ本のご紹介

### 達人プログラマーシステム開発の職人から名匠への道

経験に裏打ちされたヒントを満載し、実践可能なプログラムの作り方を教える。より効率的、そして生産的なプログラマーを目指す人に向け、具体的なアドバイスを、解決策のシステムを構成するよう関連付けて編集。

### 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。

### コーディングを支える技術～成り立ちから学ぶプログラミング作法

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。そして、今後生まれてくる新しい概念も、よりいっそう理解しやすくなることでしょう。

### 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 (標準プログラマーズライブラリ)

どうしてポインタのところでわからなくなっちゃうんだろう?なぜCのポインタはこれほどまでに難しいといわれてしまうのか—その理由は、Cの宣言まわりの混乱した文法と、ポインタと配列の間の妙な交換性にあった。Cの構文を解き明かす。C言語を勉強してみたけどポインタでつまずいた人、普段Cを使っているが実は理解が曖昧な人へ。

## 定本 Cプログラマのためのアルゴリズムとデータ構造

すべてのCプログラマに最適なアルゴリズム解説書。良いプログラムを書くためにはアルゴリズムとデータ構造の知識は必須である。本書は、C言語の初心者・中級者を対象に、アルゴリズムとデータ構造の基礎から、各アルゴリズムの特長、C言語による実装までを明快に解説している。

## 初めてのSQL

本書『初めてのSQL』は、SQL言語に初めて触れるプログラマを対象に、SQL言語の基本を解説する書籍です。その特徴は、実践に必要なポイントを過不足なく、コンパクトにまとめたこと。「データベースの歴史」から始まり「データベースの作成と設定」「クエリ」「フィルタリング」「複数テーブルからのデータの取得」「集合」「データの作成、変換、操作」「グループ化と集約化」「サブクエリ」「結合」「条件ロジック」「トランザクション」「インデックスと制約」などについて、丁寧に解説を行います。本書のサンプルは、MySQLを使用していますが、Oracle、SQL Serverなどでも動作するように配慮されています。各章末には練習問題を掲載し、読者の理解を助けます。SQL言語の基本を確実に身に付けたい方におすすめです。

## 初めてのプログラミング 第2版

初めてプログラミングを学ぶ入門者を対象に、プログラミングの基礎をていねいに解説するプログラミングの入門書。プログラミングとは何かを無理なく理解するために、要点をひとつひとつていねいに解説。簡単な概念から始めて、かなり高度なプログラミングの知識までを身に付けることができる。教材には、誰でもどんな環境でも気軽に使えるRubyを使い、実際に簡単なコードを書きながら理解を深めていく。

## たのしいRuby 第5版

Ruby入門書の超ロングセラー!

いちばん売れているRuby入門書の決定版。

初版から14年。改訂とともに変化しながら、ずっと読まれ続けている定番商品です。第5版では、最新のバージョンに対応。

プログラミング初心者でも読み解けるように、チュートリアル、基礎、クラス、実践とテーマを切り分けて、平易に解説。Rubyの基礎から応用までがわかる一冊。

## 資格試験について

### 基本情報技術者試験～ITエンジニアの登竜門～

ITエンジニアとしてキャリアをスタートするには、まず基本情報技術者試験から受験することをお勧めします。しっかりとした基礎を身に付けることにより、その後の応用力の幅が格段に広がります。

[https://www.jitec.ipa.go.jp/1\\_11seido/fe.html](https://www.jitec.ipa.go.jp/1_11seido/fe.html)

### C言語プログラミング能力認定試験

"C言語プログラミング能力認定試験"は、これからプログラミングを学び始める方を対象とした「3級」から、プログラマやシステム・エンジニアとして活躍されている方の保有スキルを客観的にアピールできる「1級」まで、幅広いスキルを測定しています。

<http://www.sikaku.gr.jp/js/cp/>

試験内容

出題範囲

受験料と試験日程

受験申込についてはスクプラの担当者に申し出て、自分が受験する等級の該当受験料を申し込んでください。サーティファイのサイトから直接ネットで申し込みすることも可能です。

- 申込前に等級ごとに過去問で腕試しをすることをお勧めします。
- 講座の授業進行状況やあなたの努力次第になりますが、1級取得を目指して頑張りましょう！

## 講座

- 初めてのプログラム
- 繰り返し処理
- 条件分岐・配列
- ファイルの取り扱い・文字の検索
- ポインタ
- 棒取りゲーム
- 構造体
- マージソート
- 在庫管理システム
- データベース
- 補足 その1
- 補足 その2
- オブジェクト指向

# 初めてのプログラム

Hello World!! プログラムの扉を開きましょう。  
ようこそ、世界！！

hello.c

```
#include <stdio.h>

main(){
    printf("Hello World!!!");
}
```

じゃんけんゲーム

rock\_paper\_scissors.c

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>

int main(int argc, char const *argv[]) {
    /* code */
    int computer;

    srand(time(NULL));
    computer = rand() % 3 + 1;
    printf("コンピュータは %d: ", computer);

    if (computer == 1){
        printf("グー");
    } else if (computer == 2){
        printf("チョキ");
    } else {
        printf("パー");
    }

    return 0;
}
```

## 繰り返し処理

### プログラムを構成する3つの要素

- 逐次・順次
- 反復・繰り返し
- 判断・条件分岐

C言語には反復のために、3つの構文が用意されています。

- for文
- while文
- do-while文

while文が基本です。

for文 繰返し回数が分かっている時 while文 繰返し回数が不明の時 (前判定) do-while文  
(後判定)

#### loop.c

```
#include <stdio.h>

main(){
    int i;

    for (i = 0; i < 3; i++) {
        printf("%d Hello\n", i);
    }

    i = 0;
    while (i < 3) {
        printf("%d Hello\n", i);
        i++;
    }

    i = 0;
    do {
        printf("%d Hello\n", i);
        i++;
    } while (i < 3);

}
```

## break / continue

break\_continue.c

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>

int main(int argc, char const *argv[]) {
    int player;
    int computer;
    int i;
    char c;

    while(1){
        printf("グーチョキパーを入れてください\n");
        scanf("%d", &player);
        while(getchar() != '\n');

        while(1){
            c = getchar();
            printf ("c: %c\n", c);
            if (c == '\n'){
                break;
            }
        }

        while(getchar() != '\n'){
            ; // 読み飛ばす
        }

        while(getchar() != '\n');

        printf("player: %d\n", player);

        if (1 <= player && player <= 3){
            printf("正しい入力です\n");
            break;
        }

        if (player < 1 || player > 3){
            printf("もう一度、入力してください\n");
            continue;
        }
    }
}
```

```
printf("while文を抜けました\n");
return 0;

for (i = 0; i < 3; i++){
    srand(time(NULL));
    computer = rand() % 3 + 1;
    printf("コンピュータは %d: ", computer);

    if (computer == 1){
        printf("グー");
    } else if (computer == 2){
        printf("チョキ");
    } else {
        printf("パー");
    }
    printf("\n");
}

return 0;
}
```

# 条件分岐・配列

## 条件分岐

条件分岐のための構文として、

- if 文
- switch 文が用意されています。

### vegetable.c

```
#include <stdio.h>

int main(int argc, char const *argv[]) {
    int vegetable;          // 野菜を食べた量
    int total = 0;           // 今まで食べた量の合計
    int ratio;               // 率
    int goal = 350;          // 目標

    int i;                  // 何回目の食事か？

    float a = 1.0;
    float b = 7.0;

    double x = 1.0;
    double y = 7.0;

    printf("%20.15f \n", a / b);
    printf("%20.15f \n", x / y);

    printf("野菜を一日350g食べましょう\n");

    for(i = 0; i < 3; i++){
        switch(i){
            case 0:
                printf("朝ごはん");
                break;
            case 1:
                printf("昼ごはん");
                break;
            case 2:
                printf("夜ごはん");
                break;
        }
    }
}
```

```
        break;
    }

    printf("何グラム食べましたか?\n");
    scanf("%d", &vegetable);
    while(getchar() != '\n');

    // 350      100      250
    total = total + vegetable;
    // 10000 / 350 --> 30%
    ratio = 100 * total / goal;
    printf("達成率は %d %% です\n", ratio); // 30%
    printf("残り %d グラムの野菜を食べましょう\n", goal - total);
}

if (total >= goal) {
    printf("\n健康な食生活です(*^_^*)\n");
} else {
    printf("\nもう少し野菜を食べましょう(*^_^*)\n");
}
}
```

## 配列

配列は、基本的なデータ構造です。

いくつもの変数を纏めて取り扱えると、繰り返し処理などで活用することが出来、とっても便利です。

- 一次元配列

[score1.c](#)

```
#include <stdio.h>

int main(int argc, char const *argv[]) {

    int student[10] = { 18, 25, 73, 30, 55, 18, 21, 89, 17, 31 };
                    // 10人の生徒の成績を格納する配列
    int top;        // 最高点
    int total = 0;   // 合計点
    int average;    // 平均点
    int i;

    // 平均点を求める処理
    total = 0;
    for(i = 0; i < 10; i++){
        for(i=0;i<10;i++){
            // 合計点を求める
            total = total + student[i];
        }
        average = total / 10;
        printf("平均点は %d です\n", average);

        // 最高点を求める処理
        top = 0;
        for (i = 0; i < 10; i++){
            if (top < student[i]){
                top = student[i];
            }
        }
        printf("最高点は %d です\n", top);

    return 0;
}
```

- 二次元配列

score2.c

```
#include <stdio.h>

int main(int argc, char const *argv[]) {

    int student[2][10] = { { 18, 25, 73, 30, 55, 18, 21, 89, 17, 31 },
                          { 28, 29, 33, 38, 15, 38, 51, 19, 77, 68 } };
    // 10人の生徒の成績を格納する配列

    int top;           // 最高点
    int total;         // 合計点
    int average;       // 最低点
    int subject, i;

    // 平均点を求める処理
    for (subject = 0; subject < 2; subject++){
        total = 0;
        for(i = 0; i < 10; i++){
            // 合計点を求める
            total = total + student[subject][i];
        }
        average = total / 10;
        printf("平均点は %d です\n", average);
    }

    // 最高点を求める処理
    for (subject = 0; subject < 2; subject++){
        top = 0;
        for (i = 0; i < 10; i++){
            if (top < student[subject][i]){
                top = student[subject][i];
            }
        }
        printf("最高点は %d です\n", top);
    }

    return 0;
}
```

## 文字列

C 言語では、文字列は、文字の配列として扱います。

string.c

```
#include <stdio.h>
#include <string.h>

int main(int argc, char const *argv[]) {

// string subject_0 = "Japanease";
char subject_0[] = "Japanease";

char subject_1[] = "Mathematics";

// string subject[] = { "Japanease", "Mathematics", "Science" };
char subject[][20] = { "Japanease", "Mathematics", "Science" };
int i;

printf("%s\n", subject_0);
printf("%s\n", subject_1);
printf("---\n");
printf("%s\n", subject[0]);
printf("%s\n", subject[1]);
printf("---\n");
for (i = 0; i < 2; i++){
    printf("%s\n", subject[i]);
}

return 0;
}
```

# ファイルの取り扱い・文字の検索

ファイルの各種モード

モード	文字列	補足説明
読み取りモード	"r"	ファイルが存在しなければエラー
書き込みモード	"w"	ファイルが存在しなければ新規作成。ファイルが存在すれば上書き
追記モード	"a"	ファイルが存在しなければ新規作成。ファイルが存在すれば追記

## 宣言など

file\_handling.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <fcntl.h> // ファイルの存在確認

int main(int argc, char const *argv[]) {

    FILE *fp; // ファイルポインタ
    char buffer[256];
    char filename[256];
    char c;
```

## ファイルの読み込み

file\_handling.c

```
///////////
// ファイル読み込み
// (一文字ずつ読み込む例)
///////////

// ファイル名の設定
strcpy(filename, "sample.txt");
// 指定されたファイルを、読み取り専用モード("r")で開く
if ( (fp=fopen(filename, "r")) == NULL ){
    printf("指定されたファイル %s を開けませんでした。\\n", filename);
    exit(1); // エラーコード1として、プログラム終了
}

// ファイルから一文字ずつ読み込む
while ( (c=getc(fp)) != EOF ){
    // 読み込んだ文字、c を画面に出力する
    printf("%c", c);
}

// ファイルを閉じる
fclose(fp);

///////////
// ファイル読み込み
// (一行ずつ読み込む例)
///////////

// ファイル名の設定
strcpy(filename, "sample.txt");
// 指定されたファイルを、読み取り専用モード("r")で開く
if ( (fp=fopen(filename, "r")) == NULL ){
    printf("指定されたファイル %s を開けませんでした。\\n", filename);
    exit(1); // エラーコード1として、プログラム終了
}

// ファイルから一行ずつ読み込む
while( fgets(buffer, sizeof(buffer), fp) != NULL ) {
    // 読み込んだ文字「列」、buffer を画面に出力する
    printf("%s", buffer);
}

// ファイルを閉じる
fclose(fp);
```

## ファイルの書き込み

file\_handling.c

```
//////////  
// ファイル書き込み  
//////////  
  
// ファイル名の設定  
strcpy(filename, "sample.txt");  
// 指定されたファイルを、書き込みモード("w")で開く  
if ( (fp=fopen(filename, "w")) == NULL ){  
    printf("指定されたファイル %s を開けませんでした。\\n", filename);  
    exit(1); // エラーコード1として、プログラム終了  
}  
  
// 一文字ずつ書き出す  
fputc('a', fp); // 直接一文字出力  
fputc('b', fp);  
c = 'c'; // 変数に代入してから出力  
fputc(c, fp);  
c = '\\n'; // 改行文字  
fputc(c, fp);  
  
// 一行ずつ書き出す  
fputs("123\\n", fp); // 直接一行出力  
char str[256] = "xyz\\n"; // 変数に代入してから出力  
fputs(str, fp);  
  
// 書式指定  
double area = 5 * 5 * 3.14; // 円の面積  
// 書式設定して、画面に出力  
printf("円の面積は %5.2f です\\n", area);  
// 書式設定して、ファイルに出力  
fprintf(fp, "円の面積は %5.2f です\\n", area);  
// 書式設定して、文字列に出力  
sprintf(str, "円の面積は %5.2f です\\n", area);  
printf("strの中身は %s です\\n", str); // 確認  
  
// ファイルを閉じる  
fclose(fp);
```

得点ファイルを読み込んで、合計点・平均点を求める

[file\\_score.c](#)

```
#include <stdio.h>
#include <stdlib.h>

#define STRING_MAX 256           // 定数の宣言

int main(int argc, char const *argv[]) {
    FILE    *fpr, *fpw;        // ファイルポインタ
    char    buffer[STRING_MAX]; // 読み込み文字のバッファ
    int     score[5];          // 読み込んだ点数を格納する配列
    int     i;                 // 配列の添え字(index)
    int     number;            // 数
    int     total;             // 合計点
    double  average;          // 平均点

    // ファイル名
    printf("score.txtファイルから得点を読み込みます。\\n");

    // 読み込みファイルオープン
    if ( (fpr=fopen("score.txt", "r")) == NULL) {
        printf("score.txt が見つかりませんでした。");
        exit(1);
    }

    // 得点ファイルからの読み込み処理
    i = 0;
    // 文字列配列bufferへ
    // STRING_MAX 文字
    // 読み取り用のファイルポインタ fpr
    // から、文字列を取得します。
    while( fgets(buffer, STRING_MAX, fpr) != NULL) {
        number = atoi(buffer); // buffer に読み取った文字列を整数に変換
        score[i] = number;
        i++;
    }

    // 読み取り用ファイルを閉じます。
    fclose(fpr);

    // 読み込んでいるか、確認
    printf("score.txtからの読み込み結果\\n");
    for (i = 0; i < 5; i++){
        printf("score[%d]: %d\\n", i, score[i]);
    }
}
```

```

// 合計点・平均点の算出
total = 0;
for (i = 0; i < 5; i++){
    total = total + score[i];
    // total += score[i];
}
// キャスト演算 (double) で
// total を一時的に
// double型として扱うよう 変換します
average = (double)total / 5;

// 書き込みファイルオープン
if ( (fpw=fopen("result.txt", "w")) == NULL) {
    printf("result.txt へ書き込めませんでした。");
    exit(1);
}

// 処理結果を、ファイルへ出力する
printf("合計は %3d 点です。\\n", total);
printf("平均は %5.2f 点です。\\n", average);

printf("result.txtファイルへ結果を書き込みました。\\n");
fprintf(fpw, "合計は %3d 点です。\\n", total);
fprintf(fpw, "平均は %5.2f 点です。\\n", average);

fclose(fpw);

}

```

## 文字の検索

詳細なアルゴリズム解説付

[string\\_find.c](#)

```

*****
*
*
* 文字列の検索
*
*****
/

```

```

#include <stdio.h>
#include <string.h>

int main(int argc, char const *argv[]) {

    // 検索対象文字列 を "shirayukihime" で初期化
    // 一文字ずつ、初期値をセットする。
    // 文字列として扱えるよう、最後には終端文字'\0'もセット。
    char princess[20] = { 's', 'h', 'i', 'r', 'a', 'y', 'u', 'k', 'i',
                          'h', 'i', 'm', 'e', '\0' };
    // 長いので、宣言と同時に初期値を与える場合に限り、次のように書けます。
    char princess[20] = "shirayukihime"; // '\0' 不要

    // 変数宣言と代入を別々にすると、エラーとなります。
    // char princess[20]; // この行はOKです。
    // princess[20] = "shirayukihime"; // この行でエラーになります。

    // 変数宣言後に、文字列をセットする場合には、
    // 文字列コピー関数 strcpy を使います。
    // char princess[20]; // この行はOKです。
    // strcpy(princess, "shirayukihime"); // このように使います。

    // 次のように一文字ずつ代入することも出来ます。
    // princess[0] = 's';
    // princess[1] = 'h';
    // princess[2] = 'i';
    // // (略)
    // princess[11] = 'm';
    // princess[12] = 'e';
    // princess[13] = '\0';

    // C 言語では
    // (他言語では、文字列型も用意されているが)
    // 文字列は、文字の配列として、表現する。

    printf("変数princess に初期値が設定されているか確認\n");
    printf("%c\n", princess[0]); // %c は 「文字」を出力する書式指定子です。
    printf("%c\n", princess[1]); // C 言語の配列は、0から始まります。
    printf("%c\n", princess[2]);
    printf("略\n");
    printf("%c\n", princess[12]); // 最後12まで出せば、確認出来ます。

    // [0], [1], …, [12]と、配列の「添字」が変化していくので、
    // 繰り返しのためのfor文を使うと簡単に書けます。
    int i;
}

```

```

for (i = 0; i <= 12; i++){
    printf("%d %c\n", i, princess[i]); // %d 整数型の書式指定子
}

printf("%s\n", princess); // %s 文字「列」型の書式指定子


// 検索の考え方
// 検索対象文字列"shirayukihime"の先頭(0文字目)から、
// 一文字ずつ、検索文字列"hime"一致するかどうかを比較します。
// shirayukihime
// ^^^^
// |||
// vvvv
// hime
if (princess[0] == 'h'){ printf("0文字目が一致しました。\\n"); }
if (princess[1] == 'i'){ printf("1文字目が一致しました。\\n"); }
if (princess[2] == 'm'){ printf("2文字目が一致しました。\\n"); }
if (princess[3] == 'e'){ printf("3文字目が一致しました。\\n"); }
// 0 文字目から 3 文字目まで 4文字が一致すれば、
// "hime" が見つかったことになります。

// 引き続いて、1文字目から
// 一文字ずつ、検索文字列"hime"一致するかどうかを比較します。
// shirayukihime
// ^^^^
// |||
// vvvv
// hime
if (princess[1] == 'h'){ printf("1文字目が一致しました。\\n"); }
if (princess[2] == 'i'){ printf("2文字目が一致しました。\\n"); }
if (princess[3] == 'm'){ printf("3文字目が一致しました。\\n"); }
if (princess[4] == 'e'){ printf("4文字目が一致しました。\\n"); }
// 1 文字目から 4 文字目まで 4文字が一致すれば、
// "hime" が見つかったことになります。

// 同様に繰り返して、9文字目から
// 一文字ずつ、検索文字列"hime"一致するかどうかを比較します。
// shirayukihime
//         ^^^^
//         |||
//         vvvv
//         hime
if (princess[ 9] == 'h'){ printf("9文字目が一致しました。\\n"); }
if (princess[10] == 'i'){ printf("10文字目が一致しました。\\n"); }

```

```

if (princess[11] == 'm'){    printf("11文字目が一致しました。\\n"); }
if (princess[12] == 'e'){    printf("12文字目が一致しました。\\n"); }
// 9 文字目から 12 文字目まで 4文字が一致すれば、
// "hime" が見つかったことになります。

// 同じ処理を繰り返しているので、
// for 文で書けば簡単に書けそうです。
int match;
for (i = 0; i <= 9; i++){
    match = 0;           // 何文字まで検索文字列と一致しているか？
    if (princess[i+0] == 'h'){ match++; }
    if (princess[i+1] == 'i'){ match++; }
    if (princess[i+2] == 'm'){ match++; }
    if (princess[i+3] == 'e'){ match++; }
    if (match == 4){
        printf("%d 文字目から hime が見つかりました。\\n", i);
    }
}

// そして、
// if (princess[i+0] == 'h'){ match++; }
// if (princess[i+1] == 'i'){ match++; }
// if (princess[i+2] == 'm'){ match++; }
// if (princess[i+3] == 'e'){ match++; }
// これらのif文も同じ処理をしていますから、
// for文で纏めて書けそうです。
keyword[0] = 'h';
keyword[1] = 'i';
keyword[2] = 'm';
keyword[3] = 'e';
// とすると
if (princess[i+0] == keyword[0]){ match++; }
if (princess[i+1] == keyword[1]){ match++; }
if (princess[i+2] == keyword[2]){ match++; }
if (princess[i+3] == keyword[3]){ match++; }
// と書き直せます。

// よって、
for (k = 0; i < 4; k++){
    if (princess[i+k] == keyword[k]){ match++; }
}
// と書くことが出来ます。

char keyword[10] = "hime";           // 検索文字列
int k;                                // 検索文字列用のindex変数

```

```

printf("変数keywordに初期値が設定されているか確認\n");
printf("keyword[0]: %c\n", keyword[0]);
printf("keyword[1]: %c\n", keyword[1]);
printf("keyword[2]: %c\n", keyword[2]);
printf("keyword[3]: %c\n", keyword[3]);

printf("変数keywordに初期値が設定されているかfor文で確認\n");
for (k = 0; k < 4; k++){
    printf("%c\n", keyword[k]);
}

for (i = 0; i <= 9; i++){
    int match = 0; // 何文字まで検索文字列と一致しているか？
    for (k = 0; k < 4; k++){
        if (princess[i+k] == keyword[k]){ match++; }
    }
    if (match == 4){
        printf("%d 文字目から hime が見つかりました。 \n", i);
    }
}
}

// 検索対象文字列、検索文字列が何文字であっても動くように、
// 文字列の長さを格納する変数を用意すれば、
// 検索プログラムの完成です。

printf("完成版のプログラム\n");
int length; // 検索対象文字列の長さ
length = strlen(princess); // strlen関数を用いると長さが取得出来る
// length => 13
int keyword_length = strlen(keyword); // 宣言と同時に関数を使って初期値を
セットすることも出来ます。
// keyword_length => 4

for (i = 0; i <= length - keyword_length; i++){
    int match = 0; // 何文字まで検索文字列と一致しているか？
    for (k = 0; k < keyword_length; k++){
        if (princess[i+k] == keyword[k]){
            match++;
        }
    }
    if (match == keyword_length){
        printf("%d 文字目から %s が見つかりました。 \n", i, keyword);
    }
}

```

```
    }  
}  
  
return 0;  
}
```

関数化

| [string\\_find2.c](#)

```
*****
*
*
* 文字列の検索
* version 1.0
* auther : taro
* date: 2016/05/15
*
*****
/
#include <stdio.h>
#include <string.h>

// string中にkeywordが含まれていれば、何文字目から始まるかを、
// 含まれていなければ、-1 を返す
int find_keyword(char const *keyword, char const *princess){
    int i, k;
    int length      = strlen(princess);
    int keyword_length = strlen(keyword);

    for (i = 0; i <= length - keyword_length; i++){
        int match = 0;// 何文字まで検索文字列と一致しているか？
        for (k = 0; k < keyword_length; k++){
            if (princess[i+k] == keyword[k]){
                match++;
            }
        }
        if (match == keyword_length){
            return i;
        }
    }
    return -1;
}

int main(int argc, char const *argv[]) {
    char princess[20] = "しらゆきひめ";
    char keyword[10] = "しらゆき";

    printf("%d\n", find_keyword(keyword, princess));

    return 0;
}
```

## 文字の置換

string\_replace.c

```
*****
*
*
* 文字列の置換
* (mononokehime -> kaguyahime)
*
*****
/
#include <stdio.h>
#include <string.h>

int main(int argc, char const *argv[]) {

    // 変数宣言
    char princess[20] = "kaguyahime";           // 検索対象文字列
    int i;                                     // 検索対象文字列用のindex変数
    int length = strlen(princess);             // 検索対象文字列の長さ
    printf("置換前の姫の名は %s です。\\n", princess);

    int match;                                // 何文字まで検索文字列と一致している
    //?
    char keyword[10] = "kaguya";                // 検索文字列
    int k;                                     // 検索文字列用のindex変数
    int keyword_length = strlen(keyword);       // 検索文字列の長さ

    char replace_keyword[10] = "mononoke";      // 置換文字列
    int r;                                     // 置換文字列用のindex変数
    int replace_keyword_length = strlen(replace_keyword); // 置換文字列の長さ

    // 文字列検索プログラム
    for (i = 0; i <= length - keyword_length; i++){
        int match = 0;
        for (k = 0; k < keyword_length; k++){
            if (princess[i+k] == keyword[k]){
                match++;
            }
        }
        if (match == keyword_length){
            printf("%d 文字目から %s が見つかりました。\\n", i, keyword);
        }
    }
}
```

```

// 01234567
// kaguyahime // princess
// ^^^^^^
// |||||
// vvvvvv
// mononoke      // keyword
// 012345
// mononoke      // replace_keyword

// 0文字目(i文字目)から一致したことが分かったので、
// mononokehime の 0 文字目から 7 文字目を
// kaguya に置き換えればよい
int find = i; // 何文字目で見つかったか保管
for (r = 0; r < replace_keyword_length; r++){
    princess[find + r] = replace_keyword[r];
}

// princess =>
// 012345678901
// kaguyakehime
// となっている。

// mononoke 8 文字
// kaguya   6 文字
// なので、2 文字 後ろの文字を前に詰めるとよい
// princess[6] = princess[8];    // 'h'
// princess[7] = princess[9];    // 'i'
// princess[8] = princess[10];   // 'm'
// princess[9] = princess[11];   // 'e'
// princess[10] = princess[12];  // '\0'; // 終端文字

int left_shift = keyword_length - replace_keyword_length; // 2 文字
左へ
for (r = find + replace_keyword_length; princess[r] != '\0'; r++) {
    princess[r] = princess[r + left_shift];
}
princess[r] = '\0';
}

// 置換結果表示
printf("%s を %s に置換しました。\\n", keyword, replace_keyword);
printf("置換後の姫の名は %s です。\\n", princess);

return 0;

```

{}

## ファイルでの文字の検索

file\_find.c

```
/*
*
*
* ファイル内に指定された文字列があるか、検索
* 検索文字列が見つかった行番号を表示する
*
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// 記号定数の定義
#define TRUE 1
#define FALSE 0

#define SIZE 1024

int main(int argc, char const *argv[]) {

    // 変数宣言
    FILE *fp;           // ファイルポインタ
    char filename[SIZE]; // ファイル名
    char keyword[128];  // 検索文字列
    char c;
    int keyword_length; // 検索文字列の長さ
    int i;              // 検索文字列中の走査位置
    int search;         // 検索文字列内を走査中かどうかのフラグ
    int count;          // 検索文字列の出現回数
    int line;           // 現在、検索中の行数
    int find_line[100] = {}; // 何行目に見つかったか
                          // = {} で全てを0に初期化出来る
                          // for(i = 0; i < 100; i++){
                          //   find_line[i] = 0;
                          // }
                          // というfor文と同じ
```

```

// 検索対象ファイル名と、検索文字列の取得
printf("ファイル内に指定された文字列があるか、検索します。\\n");
printf("検索対象ファイル名を入力して下さい。\\n");
scanf("%s", filename);
printf("検索文字列を入力して下さい。\\n");
scanf("%s", keyword);
keyword_length = strlen(keyword);

// 指定されたファイルを、読み取り専用モード("r")で開く
if ( (fp=fopen(filename, "r")) == NULL ){
    printf("指定されたファイル %s を開けませんでした。\\n", filename);
    exit(1); // エラーコード1として、プログラム終了
}

// 初期化処理
search = 1; // 検索中フラグ
i      = 0;   // 検索文字列中の走査位置
count  = 0;   // 検索文字列の出現回数
line   = 1;   // 現在、検索中の行数

// ファイル読み込み
while ( (c=fgetc(c, fp)) != EOF end of file){
    // 現在の行数の把握
    if (c == '\\n'){
        line++;
    }
}
printf("lineは%d行でした", line);

// 検索処理
if (search == 0) { // 検索文字列内を走査中の場合
    if (search) { // 検索文字列内を走査中の場合
        if (c == keyword[i]){
            // ファイルから読み取った文字 c と
            // 検索文字列の i 番目の文字が一致した。
            i++;
        } else {
            // 検索文字列に一致しない文字が現れた
            search = FALSE;
            i      = 0;
        }
    } else { // 走査中でない場合
        if (c == keyword[0]){ // キーワードの先頭文字
            search = TRUE; // 文字列の走査モードに入る
            i++;
        }
    }
}

```

```
        }

    }

    // 検索文字列と全て一致した場合
    if (i == keyword_length){
        find_line[count] = line; // 出現した行番号を格納
        count++;                // 検索文字列の出現回数
        i = 0;                  // 検索文字列中の走査位置
        search = FALSE;
    }
}

fclose(fp); // ファイルを閉じる

// 検索結果の表示
printf("指定ファイル %s 内で\n", filename);
printf("検索文字列 %s を検索しました。 \n", keyword);
printf("\n");
printf("%d 回、検索文字列が出現しました。 \n", count);
printf("出現した行番号は、次のとおりです。 \n");
for (i = 0; find_line[i] != 0; i++){ // for 文の継続条件に着目して下さい
    printf("%3d ", find_line[i]);
}
printf("\n");

return 0;
}
```

## ポインタ

### C言語ポインタ完全制覇 (標準プログラマーズライブラリ)

に詳細が記載されているので、読むと良いです。

## ポインタの基礎

pointer.c

```
#include <stdio.h>

int main(int argc, char const *argv[]) {

    int a = 5;
    int* p;      // intポインタ型 変数 p を宣言

    p = &a;      // int型の変数aのアドレスを、pに代入
    printf("a の値は %d\n", a);
    printf("a のアドレスは %p\n", &a);
    printf("a のアドレスは %08x\n", &a);
    printf("p の値      は %08x\n", p);
    printf("p の値      は %d\n", p);
    printf("p の指し示している先の値は %d\n", *p);

    *p = 100;
    printf("p の指し示している先の値は %d\n", *p);

    char c = 'a';      // 文字型の変数cを宣言 初期値を設定
    char *p;           // 文字ポインタ（指示）型の変数p を宣言
    p = &c;           // 変数cのアドレスを、pに代入
                      // (p を 変数cの「ショートカット」として設定
    printf("c の値は %c\n", c);
    printf("p の指し示している先の値は %c\n", *p);

    char str[] = "abcd"; // 文字型の配列strの宣言と、初期値
    p = &(str[0]);
    printf("str は %s\n", str);
    printf("str[0] は %c\n", str[0]);
    printf("p は %s\n", p);

    char *ch;
    ch = &str[0];
    printf("c は %c\n", *ch);

    return 0;
}
```

pointer2.c

```
#include <stdio.h>

int main(int argc, char const *argv[]) {

    int a = 5;
    int b = 10;
    int* p; // int型のポインタ型 変数 p を宣言

    // p = &a; // int型の変数aのアドレスを、pに代入
    *p = 100;

    printf("sizeof(int) %lu\n", sizeof(int));
    printf("sizeof(int*) %lu\n", sizeof(int*));
    printf("sizeof(char) %lu\n", sizeof(char));
    printf("sizeof(char*) %lu\n", sizeof(char*));

    printf("a の値は      %d\n", a);
    printf("a のアドレスは %p\n", &a);
    printf("a のアドレスは %08x\n", &a);

    printf("p の値      は %08x\n", p);
    printf("p の値      は %d\n", *p);
    printf("p の指し示している先の値は %d\n", *p);

    *p = 100;
    printf("*p = 100;\n");
    printf("p の指し示している先の値は %d\n", *p);

    p = &b; // int型の変数bのアドレスを、pに代入
    *p = 100;
    printf("*p = 100;\n");
    printf("p の指し示している先の値は %d\n", *p);

    printf("a の値は      %d\n", a);
    printf("b の値は      %d\n", b);

    return 0;
}
```

### pointer3.c

```
#include <stdio.h>
int main(int argc, char const *argv[]) {
```

```
int number[4];
number[0] = 10;
number[1] = 20;
number[2] = 30;
number[3] = 40;
puts("");
printf("number のアドレスは%p\n", number);
printf("number[0]のアドレスは%p\n", &number[0]);
printf("number[1]のアドレスは%p\n", &number[1]);
printf("number[2]のアドレスは%p\n", &number[2]);
printf("number[3]のアドレスは%p\n", &number[3]);

int *p;
// p = number;
p = &number[0];
puts("");
printf("pの値は%p\n", p);
printf("pの指し示す先の値 *pは%d\n", *p);

p = 0x0022ff30;
puts("");
printf("pの値は%p\n", p);
printf("pの指し示す先の値 *pは%d\n", *p);

printf("p+0の指し示す先の値 *(p+0)は%d\n", *(p+0));
printf("p+1の指し示す先の値 *(p+1)は%d\n", *(p+1));
printf("p+2の指し示す先の値 *(p+2)は%d\n", *(p+2));
printf("p+3の指し示す先の値 *(p+3)は%d\n", *(p+3));

puts("");
puts("for文でも書けます");
int i;
for (i = 0; i < 4; i++){
    printf("p+%dの指し示す先の値 *(p+%d)は%d\n", i, i, *(p+i));
}

puts("");
printf("p++する前のpの値 は%p\n", p);
printf("pの指し示す先の値 *pは%d\n", *p);
p++;
printf("p++ した後のpの値 は%p\n", p);
printf("pの指し示す先の値 *pは%d\n", *p);

//
```

```
// char string[4];
// string[0] = 'a';
// string[1] = 'b';
// string[2] = 'c';
// string[3] = 'd';
//
// puts("");
// printf("string のアドレスは%p\n", string);
// printf("string[0]のアドレスは%p\n", &string[0]);
// printf("string[1]のアドレスは%p\n", &string[1]);
// printf("string[2]のアドレスは%p\n", &string[2]);
// printf("string[3]のアドレスは%p\n", &string[3]);
//
// float f[4] = { 1.1, 2.2, 3.3, 4.4 };
// puts("");
// printf("f のアドレスは%p\n", f);
// printf("f[0]のアドレスは%p\n", &f[0]);
// printf("f[1]のアドレスは%p\n", &f[1]);
// printf("f[2]のアドレスは%p\n", &f[2]);
// printf("f[3]のアドレスは%p\n", &f[3]);
//
// double d[4] = { 1.1, 2.2, 3.3, 4.4 };
// puts("");
// printf("d のアドレスは%p\n", d);
// printf("d[0]のアドレスは%p\n", &d[0]);
// printf("d[1]のアドレスは%p\n", &d[1]);
// printf("d[2]のアドレスは%p\n", &d[2]);
// printf("d[3]のアドレスは%p\n", &d[3]);
//
return 0;
}
```

### pointer4.c

```
#include <stdio.h>
int main(int argc, char const *argv[]) {

    char str[5] = "test";
    char *p;

    puts("");
    printf("str のアドレスは%p\n", str);
```

```
printf("str          は%s\n", str);
printf("str[0]のアドレスは%p\n", &str[0]);
printf("str[1]のアドレスは%p\n", &str[1]);
printf("str[2]のアドレスは%p\n", &str[2]);
printf("str[3]のアドレスは%p\n", &str[3]);
printf("str[4]のアドレスは%p\n", &str[4]);

puts("");
p = str;
printf("p    の値は%p\n", p);
printf("p    の指し示す先の値は %c\n", *(p));
printf("p+1 の指し示す先の値は %c\n", *(p+1));
printf("p+2 の指し示す先の値は %c\n", *(p+2));
printf("p+3 の指し示す先の値は %c\n", *(p+3));
printf("p+4 の指し示す先の値は %c\n", *(p+4));

puts("");
// p = &str[0];
printf("str    の指し示す先の値は %c\n", *str);
printf("str+1 の指し示す先の値は %c\n", *(str+1));
printf("str+2 の指し示す先の値は %c\n", *(str+2));
printf("str+3 の指し示す先の値は %c\n", *(str+3));
printf("str+4 の指し示す先の値は %c\n", *(str+4));

char name[5] = "taro";
printf("name    のアドレスは%p\n", name);
p = name;
printf("p    の値は%p\n", p);
printf("p    の指し示す先の値は %c\n", *(p));
printf("p+1 の指し示す先の値は %c\n", *(p+1));
printf("p+2 の指し示す先の値は %c\n", *(p+2));
printf("p+3 の指し示す先の値は %c\n", *(p+3));
printf("p+4 の指し示す先の値は %c\n", *(p+4));

puts("");
printf("str          は%s\n", str);

return 0;
}
```

## pointer5.c

```
#include <stdio.h>
int main(int argc, char const *argv[]) {
    char str[5] = "test";  char *p;
    p = str;
    printf("p の値は%p\n", p);
    printf("p の指し示す先の値は %c\n", *p);
    printf("p+1 の指し示す先の値は %c\n", *(p+1));

    printf("str の指し示す先の値は %c\n", *str);
    printf("str+1 の指し示す先の値は %c\n", *(str+1));

    char name[5] = "taro";
    printf("name のアドレスは%p\n", name);
    p = name;
    printf("p の値は%p\n", p);
    printf("p の指し示す先の値は %c\n", *p);
    printf("p+1 の指し示す先の値は %c\n", *(p+1));

    str = name;
    printf("str の値は%p\n", p);
    printf("str の指し示す先の値は %c\n", *str);
    printf("str+1 の指し示す先の値は %c\n", *(str+1));

    return 0;
}
```

## ポインタと関数

p\_and\_f.c

```
*****  
*  
* 関数とポインタ  
*  
*  
*****  
  
#include <stdio.h>  
int num1, num2;  
  
void swap(int x, int y){  
    // int org = x;  
    // x = y;  
    // y = org;  
    int work;  
    work = num1;  
    num1 = num2;  
    num2 = work;  
    printf("swap num1 : %d\n", num1);  
    printf("swap num2 : %d\n", num2);  
  
    // printf("swap x      : %d\n", x);  
    // printf("swap y      : %d\n", y);  
    // printf("main num1   : %d\n", num1);  
    // printf("main num2   : %d\n", num2);  
}  
  
int main(int argc, char const *argv[]) {  
    int a, b;  
  
    a = 3; b = 5;  
    num1 = 3; num2 = 5;  
    swap(a, b);  
    // printf("main a      : %d\n", a);  
    // printf("main b      : %d\n", b);  
    printf("main num1   : %d\n", num1);  
    printf("main num2   : %d\n", num2);  
  
    return 0;  
}
```

### p\_and\_f2.c

```
*****
```

```
*  
* 関数とポインタ  
*  
*  
*****/*  
  
#include <stdio.h>  
  
void swap_failure(int x, int y){  
    printf("swap_failure &x : %p\n", &x);  
    printf("swap_failure &y : %p\n", &y);  
    int work;  
    work = x;  
    x = y;  
    y = work;  
    printf("swap_failure x : %d\n", x);  
    printf("swap_failure y : %d\n", y);  
}  
  
void swap(int *x, int *y){  
    printf("swap x : %p\n", x);  
    printf("swap y : %p\n", y);  
    int work;  
    work = *x;  
    *x = *y;  
    *y = work;  
    printf("swap *x : %d\n", *x);  
    printf("swap *y : %d\n", *y);  
}  
  
int main(int argc, char const *argv[]) {  
    int a, b;  
    a = 3;  
    b = 5;  
    puts("---");  
    printf("main &a : %p\n", &a);  
    printf("main &b : %p\n", &b);  
    puts("---");  
    swap(&a, &b);  
    printf("main a : %d\n", a);  
    printf("main b : %d\n", b);  
    puts("---");  
    swap_failure(a, b);  
  
    return 0;  
}
```

}

### p\_and\_f3.c

```
/*********************************************
*
* 関数とポインタ
*
*
********************************************/

#include <stdio.h>

// void half(int num[3]){
void half(int *num){
// void swap(int *x, int *y){
    printf("half &num[0]: %p\n", &num[0]);
    num[0] = num[0] / 2;
    num[1] = num[1] / 2;
    num[2] = num[2] / 2;
    return a;
    return b;
    *x = 100;
    *y = 200;
}

int main(int argc, char const *argv[]) {
    int num[3] = { 10, 20, 30 };
    printf("main &num[0]: %p\n", &num[0]);
    half(num);
    half(&num[0]);
    swap(&a, &b);

    printf("%d\n", num[0]);
    printf("%d\n", num[1]);
    printf("%d\n", num[2]);
}
```

## ポインタのポインタ（ダブルポインタ）

[こちら](#)にも詳細な説明があります。

### double\_pointer.c

```
#include <stdio.h>
#include <string.h>

#define MAXITEM 20

int split(char *str, char const *char *mnthp[3] = { /* ポインタの配列
の宣言 */
    "January", "February", "March"
};
char **p1, **p2, **p3; /* 「ポインタのポインタ」の宣言 */
int i, j;

p1 = p2 = p3 = mnthp; /* 「ポインタのポインタ」にポインタの配列 */
/* の先頭番地を設定 */

***** 例1 *****
for (i = 0; i < 3; i++) { /* 「ポインタのポインタ」の値を変えずに */
    printf("%s\n", *(p1 + i)); /* 相対的に文字列を出力 */
}

***** 例2 *****
for (i = 0; i < 3; i++) { /* 「ポインタのポインタ」の値そのものを */
    printf("%s\n", *p2); /* 更新して絶対的に文字列を出力 */
    ++p2;
}

***** 例3 *****
for (i = 0; i < 3; i++) {
    j = 0;
    while(*(*p3 + j) != '\0') { /* 「ポインタのポインタ」を使って、 */
        printf("%c", *(*p3 + j)); /* 1文字ずつ出力する */
        j++;
    }
    printf("\n");
    ++p3;
}
```

# 棒取りゲーム

## 様々な演算子

C 言語には様々な演算子が用意されています。

### 演算子一覧表

- 演算子を用いたプログラムの例

#### bin\_game.c

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>

#define MAX      3
#define PLAYER   1
#define COMPUTER 0

// プレーヤーの取ったビンの数
int player_get(){
    int get;
    do {
        printf("何本とりますか(1-3)\n");
        scanf("%d", &get);
        while(getchar() != '\n');
    } while ( !(1 <= get && get <= MAX) );
    return get;
}

// コンピュータの取ったビンの数
int computer_get(int bin){
    if (bin <= 4){
        return bin - 1;
    } else {
        srand(time(NULL));
        return rand() % 3 + 1;
    }
}

// ビンの表示
void disp_bin(int bin, int bin_max){
```

```
int i;
for (i = 1; i <= bin; i++){
    printf("■");
}
for (i = bin + 1; i <= bin_max; i++){
    printf("□");
}
printf("\n");
}

int main(int argc, char const *argv[]) {

    // 変数宣言
    int bin_max;
    int bin;
    int player_get_bin;
    int computer_get_bin;
    int turn;

    // ビンの総数
    srand(time(NULL));
    bin_max = rand() % 5 + 20;
    bin = bin_max;

    // オープニング
    printf("ビン取りゲーム\n");
    printf("交互に1～3本のビンを取ります。\n");
    printf("最後の一本を取ると負けです\n");
    printf("ビンの本数: %d\n", bin);

    // 先攻後攻
    printf("先攻: 1 後攻: 0\n");
    scanf("%d", &turn);
    while(getchar() != '\n');
    if (turn == 1){
        turn = PLAYER;
        printf("プレーヤーの先攻です\n");
    } else {
        turn = COMPUTER;
        printf("コンピュータの先攻です\n");
    }
    printf("ゲームスタート\n");

    // ビンを交互にとる
    do {
```

```
if (turn == PLAYER){
    printf("\nプレーヤーの番です\n");
    printf("%d 本のビンが残っています\n", bin);
    disp_bin(bin, bin_max);
    player_get_bin = player_get();
    printf("%d 本のビンを取りました。.\n", player_get_bin);
    bin -= player_get_bin;
    turn = COMPUTER;
} else {
    printf("\nコンピュータの番です\n");
    printf("%d 本のビンが残っています\n", bin);
    disp_bin(bin, bin_max);
    computer_get_bin = computer_get(bin);
    printf("%d 本のビンを取りました。.\n", computer_get_bin);
    bin -= computer_get_bin;
    turn = PLAYER;
}
} while (bin > 0);
// turn = (turn == PLAYER ? COMPUTER : PLAYER);

// エンディング
if (turn == PLAYER){
    printf("\nプレーヤーの勝ちです\n");
} else {
    printf("\nコンピュータの勝ちです\n");
}

}
```

# 講座

## 構造体の基本

名前、年齢などのように異なるデータ型を、一緒に扱いたい場合があります。文字「列」型や整数型の様に、「利用者」型があったら便利です。

char型やint型は、C言語標準で用意されていた型ですが、プログラマが、新しい「型」を創造することも可能です。

既存のデータ型を組み合わせて作られた新しい「型」のことを、「構造体」と呼びます。

### structure.c

```
/*
 *
 *
 * 構造体(structure) その1
 *
 */
#include <stdio.h>
#include <string.h>

int main(int argc, char const *argv[]) {

    // 変数宣言
    char name[20];      // 利用者の名前
    int age;             // 利用者の年齢
    // 値の代入
    strcpy(name, "taro");
    age = 30;
    // 利用者情報の表示
    printf("---通常の変数での名前と年齢の表示\n");
    printf("%s さんの年齢は %d 歳です。 \n", name, age);

    // このように名前、年齢などのように異なるデータ型を、一緒に扱いたい場合があります。
    // 文字「列」型や整数型の様に、「利用者」型があったら便利です。

    // char型やint型は、C言語標準で用意されていた型ですが、
    // プログラマが、新しい「型」を創造することも可能です。
    // 既存のデータ型を組み合わせて作られた新しい「型」のことを、
    // 「構造体」と呼びます。
```

```
// 利用者型 構造体 そのものの定義
struct user {
    char name[20];
    int age;
}; // 最後に ; (セミコロン)があることに注意

// 利用者型変数 personA, personB の宣言
struct user personA, personB;
// 値の代入
// personA.name のように personB と name の間に . (ドット) があることに着目して
下さい。
// 構造体の要素 (=メンバ変数) にアクセスするには、.(ドット演算子)を用います。
// personA「の」name というように、「.」を「の」と読み替えると理解しやすいと思います
。
strcpy(personA.name, "taro");
// personA.name = "taro";// 書けない
personA.age = 30;
strcpy(personB.name, "hanako");
personB.age = 28;
// 利用者情報の表示
printf("---初めての構造体\n");
printf("%s さんの年齢は %d 歳です。 \n", personA.name, personA.age);
printf("%s さんの年齢は %d 歳です。 \n", personB.name, personB.age);

// 変数宣言の都度、struct user と書くのは、手間です。
// せっかく user 型を作成したのですから、
// user person1, person2 として使いたいものです。

// typedef (type define)キーワードの使い方
// 先の構造体の宣言との相違点に着目して下さい。
typedef struct {
    char name[20];
    int age;
} user;

// 利用者型の変数 person1, person2の宣言
user person1, person2;
// 値の代入
strcpy(person1.name, "taro");
person1.age = 30;
strcpy(person2.name, "hanako");
person2.age = 28;
// 利用者情報の表示
printf("---typedefを使って構造体を宣言した場合\n");
```

```
printf("%s さんの年齢は %d 歳です。\\n", person1.name, person1.age);
printf("%s さんの年齢は %d 歳です。\\n", person2.name, person2.age);

// 宣言と同時に初期値をセットすることも出来ます。
user person3 = { "jiro", 20 };
printf("---宣言と同時に初期値をセット\\n");
printf("%s さんの年齢は %d 歳です。\\n", person3.name, person3.age);

// 構造体の配列を作ることも出来ます。

// 構造体の配列の変数宣言
user person[3]; // user型の配列変数personを宣言
                  // 添字は0-2 まで大きさ3の配列
// 値の代入
strcpy(person[0].name, "zero");
person[0].age = 0;
strcpy(person[1].name, "ichiro");
person[1].age = 10;
strcpy(person[2].name, "jiro");
person[2].age = 20;
// 利用者情報の表示
printf("---構造体の配列\\n");
printf("%s さんの年齢は %d 歳です。\\n", person[0].name, person[0].age);
printf("%s さんの年齢は %d 歳です。\\n", person[1].name, person[1].age);
printf("%s さんの年齢は %d 歳です。\\n", person[2].name, person[2].age);

// for文を使うことも出来ます。
int i;
printf("---for文で表示\\n");
for (i = 0; i < 3; i++) {
    printf("%s さんの年齢は %d 歳です。\\n", person[i].name, person[i].age);
}

// 構造体のコピー
int num1, num2;
num1 = 10;
num2 = num1; // num2 には num1 がコピーされて 10 になります。

// 構造体も同様にコピーすることが出来ます。
person[2] = person[1];
printf("---構造体のコピー\\n");
printf("%s さんの年齢は %d 歳です。\\n", person[1].name, person[1].age);
printf("%s さんの年齢は %d 歳です。\\n", person[2].name, person[2].age);
```

```
    return 0;
}
```

**structure2.c**

```
*****
*
*
* 構造体(structure) その2
*
*****
```

```
/*
 * 構造体(structure) その2
 *
*****
```

```
#include <stdio.h>
#include <string.h>

// user型構造体の宣言
// (一般的にはmain関数の外側に書きます)
typedef struct {
    char name[20];
    int age;
} user;

// 構造型の引数を受け取る関数を作ることも出来ます。
// user型の構造体を受け取り、10歳若くして返す関数
// (引数がポインタ型であることに着目して下さい。)

/*
 * 機能：10歳若返らせる
 * 引数：user * (利用者構造体ポインタ型 を引数に取る)
 * 戻り値：なし (void)
*****
```

```
/*
 * 機能：10歳若返らせる
 * 引数：user * (利用者構造体ポインタ型 を引数に取る)
 * 戻り値：なし (void)
*****
```

```
void rejuvenate_person(user *human){
    (*human).age -= 10;
    // *human.age と書くと、意味が不明確になります。
    // human というポインタ型（参照する型・指し示す型）の変数が
    // 参照している、「実体」は、personですので、
    // personの要素ageであることを明確にするために、
    // (*human).age と書きます。

    // なお、(*human).age と書くのは手間ですので、
    // human->age -= 10;
```

```
// という 記法も用意されています。 (通常こちらを使います)

}

int main(int argc, char const *argv[]) {

    // 変数宣言
    // user型の変数personを宣言
    user person;

    // 値の代入
    strcpy(person.name, "taro");
    person.age = 30;

    // 利用者情報の表示
    printf("若返り前の %s さんの年齢は %d 歳です。\\n", person.name, person.age);

    // 10歳若返る関数を呼び出し、若返らせる
    // user型構造体の変数、person のアドレスを渡している点に着目して下さい。
    rejuvenate_person(&person);

    // 利用者情報の表示
    printf("若返り後の %s さんの年齢は %d 歳です。\\n", person.name, person.age);

    return 0;
}
```

**structure3.c**

```
*****
*
*
* 構造体(structure) その3
*
*****
*/
#include <stdio.h>
#include <string.h>

// user型構造体の宣言
// (一般的にはmain関数の外側に書きます)
typedef struct {
    char name[20];
```

```

int score[4]; // 国語、算数、理科、社会の得点
int total; // 合計点数
} student;

// 構造体の要素（メンバー）が配列である場合もあります。

/*
 * 機能：生徒構造体型の得点計算を行う
 * 引数：student * (生徒構造体ポインタ型 を引数に取る)
 * 戻り値：なし (void)
 */
void calculate_score(student *person){
    int i, sum = 0;
    for (i = 0; i < 4; i++){
        sum += person->score[i];
        // sum += (*person).score[i];
        // と書いても同じです。
    }
    person->total = sum;
    // (*person).total = sum;
    // と書いても同じです。
}

int main(int argc, char const *argv[]) {
    // 変数宣言
    student personA;

    // 値の代入
    strcpy(personA.name, "daisuke"); // 名前は大助
    personA.score[0] = 50; // 国語の点数
    personA.score[1] = 80; // 算数の点数
    personA.score[2] = 70; // 理科の点数
    personA.score[3] = 60; // 社会の点数

    // 変数宣言と同時に初期値を与えることも出来ます。
    student personB = { "hanako", { 90, 60, 50, 70 } };

    // student型構造体の変数、person のアドレスを渡している点に着目して下さい。
    // 得点を計算してくれる関数calculate_scoreに、
    // personA, B のアドレスを渡し、得点を計算させる。
    calculate_score(&personA);
    calculate_score(&personB);
}

```

```
// 得点情報の表示
printf("%s さんの合計点は %d 点です。\\n", personA.name, personA.total);
printf("%s さんの合計点は %d 点です。\\n", personB.name, personB.total);

return 0;
}
```

[structure4.c](#)

```
*****
*
*
* 構造体(structure) その4
*
*****
*/
#include <stdio.h>
#include <string.h>

// user型構造体の宣言
// (一般的にはmain関数の外側に書きます)
typedef struct {
    char name[20];
    int age;
} user;

// 構造型の戻り値を返す関数を作ることも出来ます。

*****
*
* 機能：名前と年齢を引数として受け取り、user型の構造体を返す
* 引数：char const * 文字定数ポインタ型（参照する型・指し示す型）
*      int
* 戻り値：user構造型
*****
*/
user make_person(char const *name, int age){
    user human;
    strcpy(human.name, name);
    human.age = age;
    return human;
}
```

```
int main(int argc, char const *argv[]) {  
  
    // 変数宣言  
    // 構造体型の戻り値を返す関数から  
    // 戻り値を受け取れるように、  
    // user型の変数personを宣言します。  
    user person;  
  
    // 名前と年齢を引数として受け取り、user型の構造体を返す関数を呼び出す。  
    // 結果をpersonに受け取る（代入する）  
    person = make_person("taro", 30);  
  
    // 利用者情報の表示  
    printf("%s さんの年齢は %d 歳です。\\n", person.name, person.age);  
  
    return 0;  
}
```

## 四角形型の構造体

iPhone では画面表示にrectangle型を用いたプログラミングが可能です。  
ここでは、そのイメージを示します。

[structure\\_iphone\\_rectangle.c](#)

```
*****
*
* 構造体(structure)
*
*****
```

```
#include <stdio.h>
#include <string.h>

typedef struct {
    int x0, y0, x1, y1;
    double area;
} rectangle;

double square_area(rectangle r){
    return (r.x1-r.x0) * (r.y1-r.y0);
}

int main(int argc, char const *argv[]) {
    rectangle rect[3];
    int number[3];
    rect[0].x0 = 100; rect[0].y0 = 100;
    rect[0].x1 = 500; rect[0].y1 = 800;

    rect[1].x0 = 400; rect[1].y0 = 600;
    rect[1].x1 = 800; rect[1].y1 = 1000;

    rect[2].x0 = 400; rect[2].y0 = 600;
    rect[2].x1 = 500; rect[2].y1 = 800;

    rect[0].area = square_area(rect[0]);
    printf("面積: %5.2f\n", rect[0].area);
    rect[1].area = square_area(rect[1]);
    printf("面積: %5.2f\n", rect[1].area);
    rect[2].area = square_area(rect[2]);
    printf("面積: %5.2f\n", rect[2].area);

    return 0;
}
```

## 三角形の面積 (ヘロンの公式)

[structure\\_triangle.c](#)

```
#include <stdio.h>
#include <math.h>

typedef struct {
    int a;
    int b;
    int c;
    double area;
} triangle;

int main(int argc, char const *argv[]) {

triangle t1;

printf("辺Aの長さを入れてください。\\n");
scanf("%d", &t1.a);
while(getchar() != '\\n');

printf("辺Bの長さを入れてください。\\n");
scanf("%d", &t1.b);
while(getchar() != '\\n');

printf("辺Cの長さを入れてください。\\n");
scanf("%d", &t1.c);
while(getchar() != '\\n');

// ヘロンの公式
// Triangle by Heron's formula
double S, s, a, b, c;
a = t1.a;
b = t1.b;
c = t1.c;
s = (a + b + c) / 2;
S = sqrt(s*(s-a)*(s-b)*(s-c));
t1.area = S;

printf("面積は %5.1f です\\n", t1.area);

return 0;
}
```

# マージソート

プログラム = アルゴリズム + データ構造

ここでは、データ構造として、連結リストを紹介します。また、アルゴリズムとして、マージソートを紹介します。

## 連結リスト

連結リスト（Linked list）は、最も基本的なデータ構造の1つである。一連のノードが、任意のデータフィールド群を持ち、1つか2つの参照（リンク）により次（および前）のノードを指している。連結リストの主な利点は、リスト上のノードを様々な順番で検索可能な点である。連結リストは自己参照型のデータ型であり、同じデータ型の別のノードへのリンク（またはポインタ）を含んでいる。連結リストは場所が分かっていれば、ノードの挿入や削除を定数時間で行うことができる。連結リストにはいくつかの種類があり、片方向リスト、双方向リスト、線形リスト、循環リストなどがある。（Wikipediaより）

## マージソート

マージソートは、ソートのアルゴリズムで、既に整列してある複数個の列を1個の列にマージする際に、小さいものから先に新しい列に並べれば、新しい列も整列されている、というボトムアップの分割統治法による。大きい列を多数の列に分割し、そのそれぞれをマージする作業は並列化できる。（Wikipediaより）

- 連結リスト

### linked\_list.h

```
#define NOT_FOUND -1
#define SUCCESS     1
#define FAIL        0

// リスト用構造体定義
typedef struct list {
    struct list *next;
    int          value;
} list_t;

int insert_node(list_t *list, int number){

    // 新規確保用
    list_t *p;
    // 次のノードへのポインタ
    list_t *next;
    // 先頭から順にたどってきた最後のノード
```

```

list_t *last;

// 新しいリストの領域を確保
p = (list_t*)malloc(sizeof(list_t));

// メモリ確保に失敗
if (p == NULL){
    return FAIL;
}

// 値を代入
p->value = number;
// 次の要素は末尾と分かるようにNULLを入れる。 (番兵)
p->next = NULL;

// 先頭が末尾直前のポインタになる
last = list;

// 先頭ノードから順に末尾のノードまで移動
for (next = (*list).next; next != NULL; next = next->next){
    last = next;
}

// リストを連結する。
last->next = p;

return SUCCESS;
}

int remove_node(list_t *list, int number){
list_t *p;

// 削除要素の直前のノードへのポインタ
list_t *prev;

// 最初は先頭要素の次のリストからチェックしてるので、
// 削除要素の直前の要素は先頭要素になる。
prev = list;

// リストを末尾(NULLになる)までループ
for (p = (*list).next; p != NULL; p = p->next){
    // その値があれば
    if (p->value == number){
        // 削除要素の前のリストにつなげる
        // 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
}

```

```
if (p->next != NULL){
    // 削除直前の要素につなげる
    prev->next = p->next;
    // 削除対象要素の解放
    free(p);
    return SUCCESS;
}

// 削除要素が末尾の要素だった場合の処理
// 末尾要素にNULLを入れる
prev->next = NULL;

// 削除対象要素の解放
free(p);
return SUCCESS;
}

prev = p;
}

return NOT_FOUND;
}

int show_list(list_t *list){
if ((*list).next == NULL){
    return NOT_FOUND;
}

// NULLになるまで全部表示
list_t *p;
for (p = (*list).next; p != NULL; p = p->next){
    printf("%d ", p->value);
}
printf("\n");

return SUCCESS;
}

void release_list(list_t *list){
// 次のリストのポインタ
list_t *next;
// 削除対象のポインタ
list_t *delete_node;
next = (*list).next;

// NULLになるまでループ
while(next){
    // 削除対象のポインタを保存
```

```

    delete_node = next;
    // 次のリストのポインタを取得
    next = next->next;
    free(delete_node);
}
}

```

[linked\\_list.c](#)

```

*****
*
* 連結リスト
* https://ja.wikipedia.org/wiki/連結リスト
* http://bituse.info/c/39
*
* 変数や関数に使われる必要最低限の単語を覚える
* http://oxynotes.com/?p=8679#7
*
* マージソート
* https://ja.wikipedia.org/wiki/マージソート
* http://ppp-lab.sakura.ne.jp/ProgrammingPlacePlus/algorithm/sort/007.htm
1
*
*****
*/

```

```

#include <stdio.h>
#include <stdlib.h>

#include "linked_list.h"

int main(int argc, char const *argv[]) {
    // 線形リスト宣言と、一番最初のノード生成
    list_t *list = (list_t *)malloc(sizeof(list_t));

    // このリストにあるノードは一つだけで、
    // 次の要素はまだ空なのでNULLを入れる（番兵）
    list->next = NULL;
    list->value = 0;

    char answer;
    int number;

    while (1) {

```

```

printf("何をしますか?\n 0.終了、1.追加、2.削除、3.表示\n");
scanf("%c", &answer);
while(getchar() != '\n');

switch(answer){
case '0':
    // 解放
    release_list(list);
    return 0;
case '1':
    printf("追加する値を入力して下さい> ");
    scanf("%d",&number);
    while(getchar() != '\n');
    if (insert_node(list, number) == SUCCESS){
        printf("追加しました\n");
    } else {
        printf("追加用メモリが確保出来ませんでした\n");
    }
    break;
case '2':
    printf("削除する値を入力して下さい> ");
    scanf("%d",&number);
    while(getchar() != '\n');
    if (remove_node(list, number) == SUCCESS){
        printf("削除しました\n");
    } else {
        printf("その値を持つノードは見つかりませんでした\n");
    }
    break;
case '3':
    if (show_list(list) == NOT_FOUND){
        printf("まだ何もありません\n");
    }
    break;
default:
    printf("正しい選択肢を入力して下さい> ");
    break;
}
}
}

```

- マージソート

[merge\\_sort.c](#)

```
*****  
*  
* 連結リスト  
* https://ja.wikipedia.org/wiki/連結リスト  
* http://bituse.info/c/39  
*  
* 変数や関数に使われる必要最低限の単語を覚える  
* http://oxynotes.com/?p=8679#7  
*  
* マージソート  
* https://ja.wikipedia.org/wiki/マージソート  
* http://ppp-lab.sakura.ne.jp/ProgrammingPlacePlus/algorithm/sort/007.htm  
1  
*  
*****  
*/  
  
#include <stdio.h>  
#include <stdlib.h>  
  
#include "linked_list.h"  
  
// 関数プロトタイプ宣言  
list_t *merge(list_t *a, list_t *b);  
list_t *merge_sort_recursive(list_t *list);  
void merge_sort(list_t *list);  
  
int main(int argc, char const *argv[]) {  
    // 線形リスト宣言と、一番最初のノード生成  
    list_t *list = (list_t *)malloc(sizeof(list_t));  
  
    // このリストにあるノードは一つだけで、  
    // 次の要素はまだ空なのでNULLを入れる（番兵）  
    list->next = NULL;  
    list->value = 0;  
  
    // 初期値投入  
    insert_node(list, 30);  
    insert_node(list, 20);  
    insert_node(list, 80);  
    insert_node(list, 90);  
    insert_node(list, 50);  
    insert_node(list, 10);  
    insert_node(list, 70);  
    insert_node(list, 0);
```

```

insert_node(list, 90);
insert_node(list, 20);

printf("並び替え前\n");
show_list(list);

// マージソート
merge_sort(list);

printf("並び替え後\n");
show_list(list);

// 解放
release_list(list);

return 0;
}

// マージ (併合)
list_t *merge(list_t *a, list_t *b){
// printf("merge\n");

list_t result;           // *result ではない!!
list_t *w = &result;   // work 変数

// リストをマージ
while( a != NULL && b != NULL ){
// printf("a->value: %d b->value: %d\n", a->value, b->value);

// 昇順にソートするので、小さい方の要素を結果を優先する
if(a->value <= b->value){ // == の場合は先頭を優先すると安定なソートになる
    w->next = a;
    w      = w->next;
    a      = a->next;
} else {
    w->next = b;
    w      = w->next;
    b      = b->next;
}
}

// printf("a: %p b: %p\n", a, b);

// 残っている要素を、末尾へ連結
if(a == NULL){
    w->next = b;
}

```

```
    } else {
        w->next = a;
    }
    // printf("a: %p b: %p w: %p\n", a, b, w);
    // printf("result.next: %p\n", result.next);
    return result.next;
}

// マージソート (再帰部分、昇順)
list_t *merge_sort_recursive(list_t *list){
    // printf("recursive\n");

    list_t *a;
    list_t *b;
    list_t *w; // work 変数

    // 要素が1つしかなければ終了
    if( list == NULL || list->next == NULL ){
        // printf("list one: %p\n", list);
        return list;
    }

    // 2つのリストに分割するため、リストの中心を探す
    a = list;
    b = list->next;

    // printf("a: %p b: %p\n", a, b);
    if (b != NULL){
        b = b->next;
    }

    while (b != NULL){
        a = a->next;
        b = b->next;
        if (b != NULL){
            b = b->next;
        }
    }
}

// リストを前半と後半に分離
w = a->next;
a->next = NULL;

// printf("list: %p w: %p\n", list, w);
// 前半と後半のリストをマージする
```

## マージソート

```
return merge(merge_sort_recursive(list), merge_sort_recursive(w));
}

// マージソート (昇順)
void merge_sort(list_t *list){
    list_t *result;

    if (list == NULL){
        return;
    }

    // printf("%p\n", list->next);

    // リスト全体を対象にする
    result = merge_sort_recursive(list->next);

    list->next = result;
}
```

# 在庫管理システム

## ウォーターフォール・モデル

ウォーターフォール・モデルは、ソフトウェア工学では非常に古くからある、もっともポピュラーな開発モデル。

プロジェクトによって工程の定義に差はあるが、開発プロジェクトを時系列に、「要求定義」「外部設計（概要設計）」「内部設計（詳細設計）」「開発（プログラミング）」「テスト」「運用」などの作業工程（局面、フェーズ）にトップダウンで分割する。線表（ガントチャート）を使用してこれらの工程を一度で終わらせる計画を立て進捗管理をする。原則として前工程が完了しないと次工程に進まない（設計中にプログラミングを開始するなどの並行作業は行わない）事で、前工程の成果物の品質を確保し、前工程への後戻り（手戻り）を最小限にする。ウォーターフォール・モデルの利点は、工程の進捗管理がしやすいことである。

- 在庫管理システム

### stock\_db.txt

```
100,卵,999  
110,キャベツ,50  
111,ブロッコリー,50  
201,豚ひれ肉,100  
301,ステーキ,100
```

### stock\_managing\_system.c

```
#include <stdio.h>  
#include <string.h>  
#include <stdlib.h>  
  
#define TRUE      1  
#define FALSE     0  
#define SUCCESS   1  
#define FAIL      0  
  
#define NOT_FOUND -1  
#define COLUMN_MAX 3  
#define NAME_MAX   64  
#define BUFFER_MAX 256  
  
// stock_t型 構造体
```

```
typedef struct {
    int id;           // 商品ID
    char name[NAME_MAX]; // 商品名
    int quantity;     // 数量
    int delete_flag;   // 削除フラグ(削除ならTRUE)
} stock_t;

// グローバル変数
stock_t *stock; // 後程mallocでメモリ確保
int item_count; // 取り扱っている品目数
char const *DB_NAME = "stock_db.txt"; // データベース ファイル名

// プロトタイプ宣言

// メニュー表示
void menu();
// 1-numまでの数のキー入力
int read_number(int num);
// キー入力待ち
void hit_any_key();
// 受け取った構造体のメンバーを表示する
void show_item(stock_t stock);
// 在庫一覧表示
void show_list();
// 商品ID検索
void search_item_by_id();
// 商品名検索
void search_item_by_name();
// id で検索。該当する構造体の添え字(index)を返す
int find_by_id(int id);
// 在庫数量セット
void set_quantity();
// 新商品登録
void create_new_item();
// 旧商品削除
void destroy_old_item();
// 商品名編集
void edit_item_name();
// ファイルから読み込んだレコードを、構造体に格納する
// 戻り値は、レコード数
int load_data_file(char const *filename);
// 構造体をファイルに書き出す
// 戻り値は、SUCCESS / FAIL
int save_data_file(char const *filename);
// strを区切り文字で分離、outlistに格納する
```

```

int split(char *str, char const *delimiter, char *outlist[]);
// string中にkeywordが含まれていれば、何文字目から始まるかを、
// 含まれていなければ、NOT_FOUND を返す
int find(char const *keyword, char const *princess);

// メニュー表示
void menu(){
    printf("\n");
    printf("\n### 在庫管理システム ###\n");
    printf("\n");
    printf(" 1) 商品一覧\n");
    printf(" 2) 商品IDで在庫数を検索\n");
    printf(" 3) 商品在庫数の増減\n");
    printf(" 4) 新商品の登録\n");
    printf(" 5) 旧商品の削除\n");
    printf(" 6) 商品名の編集\n");
    printf(" 7) データの保存\n");
    printf(" 8) 商品名で在庫数を検索\n");
    printf(" 9) プログラム終了\n");
    printf("\n");
    printf("メニュー選択(1-9) > ");
}

// 1-numまでの数のキー入力
int read_number(int num){
    int n;
    do {
        scanf("%d", &n);
        while(getchar() != '\n');
    } while ( !(1 <= n && n <= num) );
    return n;
}

// キー入力待ち
void hit_any_key(){
    printf("\nHit Enter key.\n");
    while(getchar() != '\n');
}

// 受け取った構造体のメンバーを表示する
void show_item(stock_t stock){
//    if (!stock[i].delete_flag){
    if (stock.delete_flag == FALSE){
        printf("%3d %-20s %3d\n", stock.id, stock.name, stock.quantity);
    }
}

```

```

}

// 在庫一覧表示
void show_list(){
    printf("\n### 在庫一覧 ###\n");
    printf("\n");
    printf(" ID 商品名           数量\n");
    int i;
    for (i = 0; i < item_count; i++){
        show_item(stock[i]);
    }
    hit_any_key();
}

// 商品ID検索
void search_item_by_id(){
    int index, id;
    printf("\n### 商品ID検索 ###\n");
    printf("商品IDを入力してください > ");
    scanf("%d", &id);
    while(getchar()!='\n');

    // index = find_by_id(id);
    // if (index != NOT_FOUND){
    if ((index = find_by_id(id)) != NOT_FOUND){
        show_item(stock[index]);
    } else {
        printf("該当するIDの商品はありませんでした。\\n");
    }
    hit_any_key();
}

// id で検索。該当する構造体の添え字(index)を返す
int find_by_id(int id){
    int i;
    for (i = 0; i < item_count; i++){
        if (stock[i].id == id && stock[i].delete_flag == FALSE){
            return i;
        }
    }
    return NOT_FOUND;
}

// 在庫数量セット
void set_quantity(){

}

```

```

int index, id, quantity;
printf("\n### 在庫数増減 ###\n");
printf("\n");
printf("商品IDを入力してください > ");
scanf("%d", &id);
while(getchar() != '\n');

if ((index = find_by_id(id)) != NOT_FOUND){
    printf("在庫数を入力してください > ");
    scanf("%d", &quantity);
    while(getchar() != '\n');
    stock[index].quantity = quantity;
} else {
    printf("該当するIDの商品はありませんでした。 \n");
}
hit_any_key();
}

// 新商品登録
void create_new_item(){
stock_t *temp;
int id, quantity;
char name[NAME_MAX];

printf("\n### 新商品登録 ###\n");
// 構造体は、0 - item_count-1までの、item_count 個 mallocでメモリ確保している
// item_count+1 個 メモリを再確保(realloc)する
temp = (stock_t*)realloc(stock, (item_count+1) * sizeof(stock_t));
// printf("stock: %p\n", stock);
// printf("temp : %p\n", temp);
if (temp != NULL){
    // printf("realloc success\n");
    // printf("temp: %p\n", temp);
    stock = temp;
    // printf("stock: %p\n", stock);

    // メモリが確保出来たので、新商品を登録する
    do {
        printf("商品IDを入力してください > ");
        scanf("%d", &id);
        while(getchar() != '\n');
    } while(find_by_id(id) != NOT_FOUND); // id 重複チェック

    printf("商品名を入力してください > ");
    scanf("%s", name);
}
}

```

```
while(getchar() != '\n');

printf("在庫数量を入力してください > ");
scanf("%d", &quantity);
while(getchar() != '\n');

stock[item_count].id = id;
strcpy(stock[item_count].name, name);
stock[item_count].quantity = quantity;
item_count++;
} else {
    printf("新商品を登録するためのメモリが確保出来ませんでした\n");
}
hit_any_key();
}

// 旧商品削除
void destroy_old_item(){
    int index, id;
    char answer;

printf("\n### 旧商品削除 ###\n");

printf("商品IDを入力してください > ");
scanf("%d", &id);
while(getchar() != '\n');

if ((index = find_by_id(id)) != NOT_FOUND){
    show_item(stock[index]);
    printf("この商品を削除しますか？(y/n) > ");
    scanf("%c", &answer);
    while(getchar() != '\n');
    if (answer == 'y' || answer == 'Y'){
        stock[index].delete_flag = TRUE;
        printf("削除しました。\\n");
    }
} else {
    printf("該当するIDの商品はありませんでした。\\n");
}

hit_any_key();
}

// 商品名編集
void edit_item_name(){
```

```
int index, id;
char name[NAME_MAX];

printf("\n### 商品名変更 ###\n");

printf("商品IDを入力してください > ");
scanf("%d", &id);
while(getchar() != '\n');

if ((index = find_by_id(id)) != NOT_FOUND){
    printf("変更後の商品名を入力してください > ");
    scanf("%s", name);
    while(getchar() != '\n');
    strcpy(stock[index].name, name);
} else {
    printf("該当するIDの商品はありませんでした。\\n");
}

hit_any_key();
}

// ファイルから読み込んだレコードを、構造体に格納する
// 戻り値は、レコード数
int load_data_file(char const *filename){
    // 変数宣言
    FILE *fp = NULL;
    char buffer[BUFFER_MAX];
    int line_count;           // 行数

    // 読み取りモードでファイルを開く
    fp = fopen(filename, "r");
    if (fp == NULL){
        printf("Can't read '%s'\\n", filename);
        exit(1);
    }

    // 行数の取得
    line_count = 0;
    while (fgets(buffer, sizeof(buffer), fp) != NULL){
        line_count++;
    }

    // メモリの動的確保
    stock = (stock_t*)malloc(line_count * sizeof(stock_t));
```

```

if (stock == NULL){
    printf("Can't allocate memory. 'stock' is NULL.\n");
    fclose(fp);
    exit(1);
}

// シーク位置を先頭に戻す
fseek(fp, 0L, SEEK_SET);

// データの読み込み
int i = 0;
char *outlist[COLUMN_MAX];
while (fgets(buffer, sizeof(buffer), fp) != NULL){
    // 一行分bufferに読み込んだので、分離して、構造体に格納する
    split(buffer, ",", outlist);
    stock[i].id = atoi(outlist[0]);           // id
    strcpy(stock[i].name, outlist[1]);         // name
    stock[i].quantity = atoi(outlist[2]);       // quantity
    stock[i].delete_flag = FALSE;              // delete_flag
    i++;
}

// ファイルクローズ
fclose(fp);

// 読み込んだ行数を返す(=構造体の要素数)
return i;
}

// 構造体をファイルに書き出す
// 戻り値は、SUCCESS / FAIL
int save_data_file(char const *filename){
    // 変数宣言
    FILE *fp = NULL;
    int i;

    // 書き込みモードでファイルを開く
    fp = fopen(filename, "w");
    if (fp == NULL){
        printf("Can't write '%s'\n", filename);
        return FAIL;
    }

    // 削除フラグが立っていないデータを書き出す
    for (i = 0; i < item_count; i++){

```

```
    if (stock[i].delete_flag == FALSE){
        fprintf(fp, "%d,%s,%d\n", stock[i].id, stock[i].name, stock[i].quantity);
    }
}

// ファイルクローズ
fclose(fp);

return SUCCESS;
}

// strを区切り文字で分離、outlistに格納する
int split(char *str, char const *delimiter, char *outlist[]){
    char *token;
    int count = 0;

    token = strtok(str, delimiter);
    while (token != NULL && count < COLUMN_MAX){
        outlist[count++] = token;
        token = strtok(NULL, delimiter);
    }

    return count;
}

// 商品ID検索
void search_item_by_name(){
    int i;
    char keyword[NAME_MAX];
    int flag = FALSE;

    printf("\n### 商品名検索 ###\n");
    printf("商品名を入力してください > ");
    scanf("%s", keyword);
    while(getchar()!='\n');

    for (i = 0; i < item_count; i++){
        if ((find(keyword, stock[i].name)) != NOT_FOUND){
            show_item(stock[i]);
            flag = TRUE;
        }
    }
    if (flag == FALSE) {
```

```
    printf("該当する商品はありませんでした。\\n");
}
hit_any_key();
}

// string中にkeywordが含まれていれば、何文字目から始まるかを、
// 含まれていなければ、NOT_FOUND を返す
int find(char const *keyword, char const *princess){
    int i, k;
    int length          = strlen(princess);
    int keyword_length = strlen(keyword);

    for (i = 0; i <= length - keyword_length; i++){
        int match = 0;// 何文字まで検索文字列と一致しているか？
        for (k = 0; k < keyword_length; k++){
            if (princess[i+k] == keyword[k]){
                match++;
            }
        }
        if (match == keyword_length){
            return i;
        }
    }
    return NOT_FOUND;
}

int main(int argc, char const *argv[]) {
    int i, count;

    // データファイルの読み込み
    item_count = load_data_file(DB_NAME);

    // メニュー表示繰り返し
    while(1){
        menu();
        switch (read_number(9)){
            case 1: show_list();           break;
            case 2: search_item_by_id();   break;
            case 3: set_quantity();        break;
            case 4: create_new_item();     break;
            case 5: destroy_old_item();   break;
            case 6: edit_item_name();      break;
            case 7:
                if(save_data_file(DB_NAME)==SUCCESS){
                    printf("データを保存しました");
                }
        }
    }
}
```

```

    };
    break;
case 8: search_item_by_name(); break;
case 9:
    printf("プログラムを終了します\n");
    // malloc で確保した領域の解放
    free(stock);
    exit(1);
    break;
}
}

return 0;
}

```

- 構造体の並び替え

#### [stock\\_sort.c](#)

```

#include <stdio.h>
#include <string.h>
#include <stdlib.h> // qsort
#include <limits.h> // 整数の取り得る範囲が定義されています。

#define TRUE 1
#define FALSE 0

//stock_t構造体
typedef struct {
    int id;          //商品ID
    char name[64-8]; //商品名
    int quantity;   //在庫数
} stock_t;

// グローバル変数
stock_t stock[11]; // 在庫構造体配列

// 表示関数
void show_item(stock_t s){
    printf("%3d %-20s %3d\n", s.id, s.name, s.quantity);
}
void show_all_item(){
    int i;
    for (i = 0; i <= 10; i++){
        printf("%3d %-20s %3d\n", stock[i].id, stock[i].name, stock[i].quantity);
    }
}

```

```

y);
}

}

void show_all_item_p(stock_t *(p[11])){
    // printf("stock[10].name: %s\n", stock[10].name);
    // printf("( * p[10] ).name : %s\n", (*p[10]).name);
    // printf("p[10]->name : %s\n", p[10]->name);
    int i;
    for (i = 0; i <= 10; i++){
        printf("%3d %-20s %3d\n", p[i]->id, p[i]->name, p[i]->quantity);
    }
}

// 設定関数
void set_stock(stock_t *s, int id, char *name, int quantity){
    (*s).id = id;
    strcpy(( * s ).name, name);
    (*s).quantity = quantity;
}

// 初期化関数
void init_stock(){
    set_stock(&stock[0], 100, "大福", 0);
    set_stock(&stock[1], 101, "いちご", 10);
    set_stock(&stock[2], 102, "にんじん", 20);
    set_stock(&stock[3], 103, "サンダル", 30);
    set_stock(&stock[4], 104, "ヨット", 40);
    set_stock(&stock[5], 105, "ごま塩", 50);
    set_stock(&stock[6], 106, "口ケット", 60);
    set_stock(&stock[7], 107, "七面鳥", 70);
    set_stock(&stock[8], 108, "蜂", 80);
    set_stock(&stock[9], 109, "くじら", 90);
    set_stock(&stock[10], 110, "ジュース", 100);
}

// 選択ソート
void selection_sort(){

    stock_t min; // 最小値格納用 構造体

    int sorted; // 今、何番目の要素まで並び替えが終了しているのか。
    int index;
    int i;
}

```

```

for (sorted = 10; sorted >= 1; sorted--){
    min.quantity = INT_MAX;
    for (i = 0; i <= sorted; i++){
        if (min.quantity > stock[i].quantity) {
            min.id      = stock[i].id;
            strcpy(min.name, stock[i].name);
            min.quantity = stock[i].quantity;
            index = i;
        }
    }
    // 一番小さい数が後ろになるように、一つずつ詰める
    for (i = index; i < sorted; i++){
        stock[i].id      = stock[i + 1].id;
        strcpy(stock[i].name, stock[i + 1].name);
        stock[i].quantity = stock[i + 1].quantity;
    }
    // 最小値をセットする
    // array[sorted] = min;
    stock[sorted].id      = min.id;
    strcpy(stock[sorted].name, min.name);
    stock[sorted].quantity = min.quantity;
}
}

void pointer_sort(stock_t *(p[11])){
    printf("pointer_sort\n");
    // 値表示 (元)
    printf("address &p[0] : %p\n", &p[0]);
    printf("address &stock[0]: %p\n", &stock[0]);
    printf("value p[0] : %p\n", p[0]);
    // 値表示 (参照先)
    printf("stock[10].name: %s\n", stock[10].name);
    printf("(p[10]).name : %s\n", (*p[10]).name);
    printf("p[10]->name : %s\n", p[10]->name);

    int min;          // 最小値
    stock_t *min_p;

    int sorted; // 今、何番目の要素まで並び替えが終了しているのか。
    int index;
    int i;

    for (sorted = 10; sorted >= 1; sorted--){
        min = INT_MAX;
        for (i = 0; i <= sorted; i++){

```

```

        if (min > p[i]->quantity){
            min = p[i]->quantity;
            min_p = p[i];
            index = i;
        }
    }

    // 一番小さい数が後ろになるように、一つずつ詰める
    for (i = index; i < sorted; i++){
        p[i]      = p[i + 1];
    }

    // 最小値をセットする
    // array[sorted] = min;
    p[sorted] = min_p;
}

}

// 比較関数（大小判断を返す関数）
int compare(const void *a, const void *b){
    // b > a なら 正の数を返す。（降順）
    return (*(stock_t *)b).quantity - (*(stock_t *)a).quantity;
}

int main(int argc, char const *argv[]) {

    /***** init *****/
    init_stock();
    // show_all_item();

    /***** selection sort *****/
    // 選択ソート
    // selection_sort();

    /***** insertion *****/
    // ポイントソート
    stock_t *(p[11]);      // ポイント配列 宣言
    // 初期化
    p[0] = &stock[0];
    p[1] = &stock[1];
    p[2] = &stock[2];
    p[3] = &stock[3];
    p[4] = &stock[4];
    p[5] = &stock[5];
    p[6] = &stock[6];
    p[7] = &stock[7];
}

```

```

p[8] = &stock[8];
p[9] = &stock[9];
p[10] = &stock[10];

printf("main\n");

printf("address &p[0]      : %p\n", &p[0]);
// 値表示 (元)
printf("address &stock[0]: %p\n", &stock[0]);
printf("value   p[0]      : %p\n", p[0]);

printf("address &p[1]      : %p\n", &p[1]);
// 値表示 (元)
printf("address &stock[1]: %p\n", &stock[1]);
printf("value   p[1]      : %p\n", p[1]);

// 値表示 (参照先)
printf("stock[10].name: %s\n", stock[10].name);
printf("( *p[10]).name : %s\n", (*p[10]).name);
printf("p[10]->name   : %s\n", p[10]->name);

pointer_sort(p);
int i;
for (i = 0; i <= 10; i++){
    printf("value   p[%d]      : %p\n", i, p[i]);
}
// ポインタは並び変わっているので、ポインタを基準に表示してみる。
// show_all_item_p(p);
// もとの構造体は並び変わっていないので、元のままの表示である。
// show_all_item();

/****************************************/
// quick sort
// 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
// for 文で 一要素ずつコピーしても良いが、
// memcpy 関数を紹介
stock_t backup[11];
memcpy(backup, stock, 11 * sizeof(stock_t)); // stock_t型構造体11要素分をbackupへコピー

// バックアップされていることの確認
// show_item(backup[10]);

// quick sort で構造体を直接並び替える
// 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

```

```
qsort(stock, 11, sizeof(stock_t), compare);

// 結果表示
show_all_item();

return 0;
}
```

# データベース

## データベース

データベース（英: database, DB）とは、検索や蓄積が容易にできるよう整理された情報の集まり。通常はコンピュータによって実現されたものを指すが、紙の住所録などをデータベースと呼ぶ場合もある。狭義には、データベース管理システム (Database Management System, DBMS) またはそれが扱う対象のことをいう。(Wikipediaより)

## CRUD

CRUD（クラッド）とは、ほとんど全てのコンピュータソフトウェアが持つ永続性[1]の4つの基本機能のイニシャルを並べた用語。その4つとは、Create（生成）、Read（読み取り）、Update（更新）、Delete（削除）である。ユーザインターフェースが備えるべき機能（情報の参照/検索/更新）を指す用語としても使われる。(Wikipediaより)

## ACID)

ACIDとは、信頼性のあるトランザクションシステムの持つべき性質として1970年代後半にジム・グレイが定義した概念で、これ以上分解してはならないという意味の原子性（英: atomicity、不可分性）、一貫性（英: consistency）、独立性（英: isolation）、および永続性（英: durability）は、トランザクション処理の信頼性を保証するために求められる性質であるとする考え方である。(Wikipediaより)

## SQLite

様々なデータベースがあります。ここでは、初心者向けに学習しやすいSQLiteをご紹介します。

SQLite入門 (<http://www.dbonline.jp/sqlite/>) で学習されて下さい。

## SQL文 サンプル

```
select * from stock;
select name from stock;
select * from stock order by quantity;
select * from stock order by quantity desc;
select * from stock where quantity > 100;
select * from stock where id >= 300 and quantity > 0;
select * from stock where id >= 300 and quantity > 0 order by quantity des
c;
select * from stock where name like "%卵%";
select * from stock where name like "卵%";
select * from stock where name like "%卵";
select * from stock;

-- insert into stock values(400, "キャベツ", 123);
-- insert into stock values(401, "白菜", 234);
select * from stock;

-- insert into stock(name, quantity) values("玉ねぎ", 345);

select * from stock where id > 400;

-- INSERT INTO stock DEFAULT VALUES;

select * from stock;

--UPDATE stock SET quantity = 50 WHERE id = 400;
-- select * from stock where id = 400;

-- UPDATE stock SET name = "愛知県産キャベツ" WHERE id = 400;
-- select * from stock where id = 400;

--UPDATE stock SET quantity = 100 WHERE id = 400;
--UPDATE stock SET quantity = 1000;

--DELETE FROM stock WHERE id=300;
DELETE FROM stock;
select * from stock;

insert into invoice(company, tel) values("田中産業", "03-xxxx-xxxx");
insert into invoice(company, tel) values("丸八青果", "06-xxxx-xxxx");
insert into invoice(company, tel) values("伊藤農園", "052-xxxx-xxxx");

select * from invoice;
```

## C言語でのデータベース操作例

### sqlite3\_sample.c

```
/*
 * sqlite3 データベースを
 * C 言語で 操作する
 *
 * 環境構築
 * http://cbbandtqb.seesaa.net/article/259881291.html
 *
 * サンプルプログラム
 * http://home.a00.itscom.net/hatada/db/sqlite/sqlite01.html
 *
 * SQLite3 ダウンロード先
 * https://www.sqlite.org/download.html
 *
 * SQLite3 を GUIで操作
 * DB Browser for SQLite
 * http://forest.watch.impress.co.jp/library/software/sqlitebrowser/
 *
 * 1週間で学ぶIT基礎の基礎ITpro 忙しいあなたのためのSQL入門
 * http://itpro.nikkeibp.co.jp/article/COLUMN/20061213/256848/
 */

#include <stdio.h>
#include <stdlib.h>
#include <sqlite3.h>

// 抽出結果が返るコールバック関数
int callback(void *not_used, int argc, char *argv[], char *column_name[]){
    int i;
    for(i = 0; i < argc; i++){
        printf("%-8s : %-8s\n", column_name[i], argv[i] ? argv[i] : "NULL");
    }
    printf("\n");
    return 0; // success
}

int main(int argc, char const *argv[]) {

    sqlite3 *db; // データベース db へのポインタを宣言
    char *error_message; // エラーメッセージ

    // データベースファイルを開く
```

```
// (もしファイルがなければ新規作成する)
int rc = sqlite3_open("stock_db.db", &db);

// // テーブル生成SQL文
// char *create_sql = "create table stock(id integer not null, name text
not null, age integer not null)";
// // テーブルを生成
// rc = sqlite3_exec(db, create_sql, 0, 0, &error_message );

// データ追加
sqlite3_exec(db, "insert into stock values(0,'いちご',10)", 0, 0, &error_
message );
sqlite3_exec(db, "insert into stock values(1,'人参',20)", 0, 0, &error_me
ssage );
sqlite3_exec(db, "insert into stock values(2,'サンダル',30)", 0, 0, &error
_message );

// stockテーブルの全項目を列挙
rc = sqlite3_exec(db, "select * from stock", callback, 0, &error_message
);

// データベースを閉じる
sqlite3_close(db);
}
```

## 補足 その 1

- 自作関数の例

add.c

```
#include <stdio.h>

// 関数のプロトタイプ（原型）宣言
int calc(int a, int b, char op);
int add(int n, int m);

int main(int argc, char const *argv[]){
    char op = '+';
    int num1 = 3;
    int num2 = 5;

    int answer;

    answer = calc(num1, num2, op);

    printf("%d\n", answer);

    return 0;
}

int calc(int a, int b, char op){
    int ans;

    if (op == '+') {
        // ans = a + b;
        ans = add(a, b);
    } else if (op == '-') {
        ans = a - b;
    } else if (op == '*') {
        ans = a * b;
    } else if (op == '/') {
        ans = a / b;
    }
}
```

```
switch(op){  
    case '+':  
        ans = a + b;  
        break;  
    case '-':  
        ans = a - b;  
        break;  
    case '*': ans = a * b; break;  
    case '/': ans = a / b; break;  
}  
  
return ans;  
}  
  
int add(int n, int m){  
    return n + m;  
}
```

- 符号付き, 符号無し, 浮動小数点数の誤差

binary.c

```
#include <stdio.h>

int main(int argc, char const *argv[]) {

    printf("EOF: %d\n", EOF);

    unsigned char uc = 0xFF; // 255
    signed char sc = 0xff; // -1

    printf("uc: %d\n", uc);
    printf("sc: %d\n", sc);

    int i;
    double dsum = 0;
    for (i = 0; i < 100; i++){
        dsum += 0.01;
    }
    printf("%.15f\n", dsum);

    float fsum = 0;
    for (i = 0; i < 100000000; i++){
        fsum += 0.0000001;
    }
    printf("%.15f\n", fsum);

    return 0;
}
```

- ファイル一覧(Mac版)

#### file\_entry.c

```
*****
*
*
* ファイル一覧
* (http://kaworu.jpn.org/kaworu/2008-05-05-1.php より引用)
*
*****
*/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
```

```

#include <err.h>
#include <sys/types.h>
#include <dirent.h>

#define TRUE      1
#define FALSE     0
#define NOT_FOUND -1

// ディレクトリエントリをリストに含めるかどうかを決めるための関数
int selects(const struct dirent *dir){
    if(dir->d_name[0] == '.'){
        return FALSE; // .(ドット)で始まるファイル名を除外する
    }
    return TRUE;
}

// 自作の並び替えの為の比較関数
int compare(const struct dirent **s1, const struct dirent **s2){
    return strcmp( (*s2)->d_name, (*s1)->d_name); // 逆順に並び替える例
}

int main(int argc, char const *argv[]){
    int i;

    const char *dirname = "./"; // ディレクトリ名
                           // (./はカレントディレクトリ)
    struct dirent **namelist; // ディレクトリにあるファイル名を格納する構造体

    // scandir関数で、ディレクトリにあるファイル名を取得する
    // 第三引数は、ディレクトリエントリをリストに含めるかどうかを決めるための関数
    // 第四引数は、並び替え関数名 alphasortはアルファベット順
    int r = scandir(dirname, &namelist, selects, alphasort);
    // 自作の並び替え用の比較関数compareを用いた例
    // int r = scandir(dirname, &namelist, selects, compare);

    if (r == NOT_FOUND){
        err(EXIT_FAILURE, "%s というディレクトリは見つかりませんでした\n", dirname);
    }
    printf("%d 個のファイルが見つかりました\n", r);
    for (i = 0; i < r; i++){
        printf("%s %d %d\n", namelist[i]->d_name, namelist[i]->d_reclen, namelist[i]->d_namlen);
        // そのディレクトリにいくつのファイルがあるかは予測不可
        // なので、namelistはmallocで取得されている
    }
}

```

```
// 使用が終わったら、freeが必要
    free(namelist[i]);
}
free(namelist);

exit(EXIT_SUCCESS);
}
```

- 文字化け対策の例

#### [moji\\_shiftjis.c](#)

```
#include <stdio.h>

int main(int argc, char const *argv[]) {
    // char eto1[] = { '90', '\', '\0' };
    char eto1[] = { "申\" };
    char eto2[] = { "酉" };
    printf("%s \n", eto1);
    printf("\\\"");
    return 0;
}
```

- メモリの動的確保

#### [realloc\\_sample.c](#)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

//stock_t構造体
typedef struct {
    int id;          //商品ID
    char name[100]; //商品名
    int quantity;   //在庫数
    int delete_flag; //削除ならTRUE
} stock_t;

stock_t *stock; //変数宣言
// stock_t stock[100];

int main(int argc, char const *argv[]) {
    stock_t *temp; // ローカル変数
```

```

stock = (stock_t *)malloc(2 * sizeof(stock_t));
if(stock != NULL) {
    stock[0].id      = 100;
    strcpy(stock[0].name, "いちご");
    stock[0].quantity = 10;

    stock[1].id      = 200;
    strcpy(stock[1].name, "にんじん");
    stock[1].quantity = 20;
}

printf("%d %s %d\n", stock[0].id, stock[0].name, stock[0].quantity);
printf("%d %s %d\n", stock[1].id, stock[1].name, stock[1].quantity);
printf("%d %s %d\n", stock[2].id, stock[2].name, stock[2].quantity);

temp = (stock_t *)realloc(stock, 3 * sizeof(stock_t));
if(temp != NULL) {
    stock = temp;
    stock[2].id      = 300;
    strcpy(stock[2].name, "サンダル");
    stock[2].quantity = 30;
}

printf("%d %s %d\n", stock[0].id, stock[0].name, stock[0].quantity);
printf("%d %s %d\n", stock[1].id, stock[1].name, stock[1].quantity);
printf("%d %s %d\n", stock[2].id, stock[2].name, stock[2].quantity);

free(stock);
return 0;
}

```

- 前置と後置

#### [prefix\\_postfix.c](#)

```

#include <stdio.h>

int main(int argc, char const *argv[]) {

    int a;
    int b;
    int w;

    a = 3; b = 5;
    w = ++b % a;
}

```

```
printf("a %d\n",w);
a = 3; b = 5;
w = --b % a;
printf("i %d\n",w);
a = 3; b = 5;
w = b++ % a;
printf("u %d\n",w);
a = 3; b = 5;
w = b-- % a;
printf("e %d\n",w);

printf("----(24)---\n");
a = 2; b = 4;
w = (b--) % (a--);
printf("a %d\n",w);
a = 2; b = 4;
w = (b--) / (a--);
printf("i %d\n",w);
a = 2; b = 4;
w = (++b) % (a++);
printf("u %d\n",w);
a = 2; b = 4;
w = (++b) / (a++);
printf("e %d\n",w);

printf("----(25)---\n");
a = 3; b = 4;
w = (++a) == b;
printf("a %d\n",w);
a = 3; b = 4;
w = a = (b -= 2);
printf("i %d\n",w);
a = 3; b = 4;
w = a >= (b -= 2);
printf("u %d\n",w);
a = 3; b = 4;
w = (a + 1) < b;
printf("e %d\n",w);

printf("----(26)---\n");
a = 4; b = 3;
w = !(a - ++b);
printf("a %d\n",w);
a = 4; b = 3;
w = !(a / b);
```

```
printf("i %d\n",w);
a = 4; b = 3;
w = !(a % b);
printf("u %d\n",w);
a = 4; b = 3;
w = !(--a / b--);
printf("e %d\n",w);

return 0;
}
```

## 補足 その2

### フローチャート

"プログラムの処理の流れ"のフローチャートの書き方

(<http://qiita.com/dokubeko/>) を参照

### ASCII コード表

ASCII文字コード

(<http://e-words.jp/p/r-ascii.html>) を参照

### 文字コードについて

Mac / Linux では、utf-8が広く用いられています。

Windowsでは、shift-jisです。

文字化けする文字一覧表

文字	コード
一	81 5C
ソ	83 5C
ゝ	84 5C
ゝ	87 5C
噂	89 5C
涅	8A 5C
欺	8B 5C
圭	8C 5C
構	8D 5C
蚕	8E 5C

```
</td>
<td width="20%">
```

```
<table cellpadding="10" cellspacing="0" border="1">
  <tbody><tr>
    <th>文字</th>
    <th>コード</th>
  </tr>

  <tr>
    <th>十</th>
    <td> 8F 5C</td>
  </tr>

  <tr>
    <th>申</th>
    <td>90 5C</td>
  </tr>

  <tr>
    <th>曾</th>
    <td>91 5C</td>
  </tr>

  <tr>
    <th>筭</th>
    <td>92 5C</td>
  </tr>

  <tr>
    <th>貼</th>
    <td>93 5C</td>
  </tr>

  <tr>
    <th>能</th>
    <td> 94 5C</td>
  </tr>

  <tr>
    <th>表</th>
    <td>95 5C</td>
  </tr>

  <tr>
    <th>暴</th>
    <td>96 5C</td>
  </tr>

  <tr>
    <th>予</th>
    <td>97 5C</td>
  </tr>

  <tr>
    <th>禄</th>
    <td>98 5C</td>
  </tr>

</tbody></table>

</td>
<td width="20%">

<table cellpadding="10" cellspacing="0" border="1">
  <tbody><tr>
    <th>文字</th>
    <th>コード</th>
  </tr>
```

```
<tr>
  <th>兔</th>
  <td> 99 5C</td>
</tr>

<tr>
  <th>喀</th>
  <td> 9A 5C</td>
</tr>

<tr>
  <th>媾 </th>
  <td> 9B 5C</td>
</tr>

<tr>
  <th>彌</th>
  <td> 9C 5C</td>
</tr>

<tr>
  <th>拿</th>
  <td>9D 5C</td>
</tr>

<tr>
  <th>朽</th>
  <td>9E 5C</td>
</tr>

<tr>
  <th>歛 </th>
  <td> 9F 5C</td>
</tr>

<tr>
  <th>濬</th>
  <td>E0 5C</td>
</tr>

<tr>
  <th>畚</th>
  <td>E1 5C</td>
</tr>

<tr>
  <th>秉</th>
  <td> E2 5C</td>
</tr>

</tbody></table>

</td>
<td width="20%">

<table cellpadding="10" cellspacing="0" border="1">
  <tbody><tr>
    <th>文字</th>
    <th>コード</th>
  </tr>

  <tr>
    <th>綵</th>
    <td> E3 5C</td>
  </tr>

```

```

<tr>
  <th>臀</th>
  <td> E4 5C</td>
</tr>

<tr>
  <th>鶩 </th>
  <td> E5 5C</td>
</tr>

<tr>
  <th>觸</th>
  <td> E6 5C</td>
</tr>

<tr>
  <th>體</th>
  <td> E7 5C</td>
</tr>

<tr>
  <th>鐸</th>
  <td>E8 5C</td>
</tr>

<tr>
  <th>饅 </th>
  <td> E9 5C</td>
</tr>

<tr>
  <th>鶴</th>
  <td>EA 5C</td>
</tr>

<tr>
  <th>僱</th>
  <td>ED 5C</td>
</tr>

<tr>
  <th>砾</th>
  <td> EE 5C</td>
</tr>

</tbody></table>

</td>

</tr>
</tbody></table>

```

<http://www.psl.ne.jp/references/0x5c.html> より引用

## 再帰

再帰（さいき）とは、あるものについて記述する際に、記述しているものそれ自身への参照が、その記述中にあらわれることをいう。定義において、再帰があらわれているものを再帰的定義という。（Wikipediaより）

## 状態遷移図

状態遷移図（じょうたいせんいす、State Transition Diagram）とは、有限オートマトンをグラフィカルに表現するために使われる図のこと。（Wikipediaより）

## 記憶クラス

変数の記憶クラス(auto, static, register, extern) 関数の記憶クラス(extern, static)

## 列挙型

列挙型（enumerated type）とは、コンピュータプログラミングにおいて、プログラマが選んだ各々の識別子をそのまま有限集合として持つ抽象データ型である。列挙型は一般に番号順を持たないカテゴリ変数(カードの組のように)として使われる。実行時には、列挙型は整数で実装されることが多い（各々の識別子は異なる整数值を持つ）。

また列挙型は、整数を使用する場合と比較して、明示的にマジックナンバーを使用するよりもプログラムソースの可読性を改善するのに役立つ。言語によっては、列挙型の整数表現はプログラマに見えないようになっていることもあり、これによりプログラマが列挙値に対して算術演算を行うような乱用を防いでいる。

言語によっては、真偽値の論理型は、あらかじめ宣言された二値の列挙型とされている。（Wikipediaより）

## 共用体

共用体（きょうようたい、union）は、プログラミング言語におけるデータ型の一つで、同じメモリ領域を複数の型が共有する構造である。

例として、ある入力が数字の場合は数値として、そうでない場合は文字列のまま保持したいという場合を考える。この場合、数値用と文字列用の領域をそれぞれ用意するのが一つの解法だが、入力は数値か文字列のどちらか一方なので、片方しか使われず無駄が出る。そこで代わりに、格納用の領域を一つだけ用意して、これを数値である、文字列であると場合により解釈し分けることで領域の無駄が抑えられる。

この「格納用の領域」こそが共用体である。（Wikipediaより）

## 開発プロセス

開発基本の基本を学ぶ

## テスト

1. テスト概論
  - テストの目的/品質との関わり
  - 一般的なテストの流れ(計画→設計→実施)
2. テストの種類
  - スコープによるテストの分類(単体・結合・システムテスト)
  - 目的によるテストの分類(機能・非機能・システムテスト)

3. テスト手法
  - ブラックボックス法
  - ホワイトボックス法
4. 代替プログラム
  - スタブとドライバ
  - スタブを使ったテスト

(<http://www.ogis-ri.co.jp/learning/c-02-00000298.html>より)

# オブジェクト指向

(Wikipediaより)

オブジェクト指向とは、オブジェクト同士の相互作用として、システムの振る舞いをとらえる考え方である。英語の *object-oriented* (直訳は、「対象物志向の」「目的重視の」という意味の形容詞) の日本語訳である。オブジェクト指向の枠組みが持つ道具立ては、一般的で強力な記述能力を持つ。複雑なシステム記述、巨大なライブラリ（特に部品間で緊密で複雑な相互関係を持つもの）の記述においては、オブジェクト指向の考え方は必須である。

## パラダイムとしてのオブジェクト指向

オブジェクト指向分析が提唱される以前には、システム分析のレベルにおいては、データ構造を中心としたシステムの分析技法である構造化技法が存在した。

また、プログラミングのレベル（プログラミングパラダイム）では、プログラムの実行の流れを決められた制御構造の組み合わせとして書き下す構造化プログラミングや、カプセル化を促すモジュールプログラミング、多態に対応するデータ指向プログラミングという技法が存在していた。オブジェクト指向手法はそれらを一般化しさらに推し進めたものであるという考え方がある。

## オブジェクト指向プログラミングの構成要件

オブジェクト指向プログラミングを構成する概念は次のようなものである。

- カプセル化（情報隠蔽）オブジェクトの振る舞いを隠蔽したり、オブジェクト内部のデータを隠蔽したり（データ隠蔽）、オブジェクトの実際の型を隠蔽したりすることをいう。これは古典的な可視性の定義である。また、オブジェクト指向プログラミングの概念拡大に伴い、必須と表現するのが不適切になりつつあるが、旧来の多くのオブジェクト指向言語が備えている性質には以下のものがある。
- ポリモーフィズム（多態性）あるオブジェクトへの操作が呼び出し側ではなく、受け手のオブジェクトによって定まる特性。クラスベースのオブジェクト指向の場合には、派生クラスの複数分岐として多態性を実現する。プロトタイプベースのオブジェクト指向の場合では関係がない概念とされる。
- オブジェクト指向の方式
  - クラスベース方式 — クラスを定義し、それを元にインスタンスを生成する方式である。継承ベースともいう。
  - プロトタイプベース方式 — 既存のインスタンスを元に、新たなインスタンスを生成する方式である。インスタンスベースともいう。

- Mixin方式 — さまざまなオブジェクトの原型を組み合わせて一つのオブジェクトを構成する方式である。

## オブジェクト指向の名称とメッセージング

Eclipseを開発したDave Thomasや、オブジェクト指向という言葉の生みの親であるAlan Kay博士は、オブジェクト指向という言葉は失敗だったと語っている。<sup>[1]</sup> これは、本来オブジェクト指向が重視すべきは「オブジェクト」ではなく「メッセージング」であるにもかかわらず「メッセージング」がおろそかにされているためである。特に言語の進歩において「オブジェクト」や「クラス」の側面ばかり強調される傾向にあり、Alan Kay博士は「Smalltalkが最高に好きという訳ではないが、他の言語に比べればマシである。」と述べている。

[オブジェクト指向プログラミング](#) (Wikipedia)も参考にして下さい。

## C言語 練習課題

★1～★7まで難易度別に練習課題を用意しました。

難易度は目安です。

挑戦してみましょう。

- 難易度 ★☆☆☆☆☆☆☆
- 難易度 ★☆☆☆☆☆☆☆
- 難易度 ★★★☆☆☆☆☆
- 難易度 ★★★★★☆☆☆
- 難易度 ★★★★★☆☆☆
- 難易度 ★★★★★★★☆
- 難易度 ★★★★★★★★

## 難易度 ★☆☆☆☆☆☆☆

1. 「おはようございます」と挨拶するプログラムを作つてみましょう。
2. 「今、何時？」と聞いて、

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つてみましょう。

3. 計算しましょう。  
 $3 + 8 \times 3$  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?
4. 高さ 29. 7 cm、幅 21. 0 cm の四角形の面積は何 cm<sup>2</sup> でしょうか？(A4 の紙一枚の大きさです)
5. 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るので、考えてみましょう。

6. 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つてみましょう。
7. 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20%なら "傘はいらないです"  
降水確率50%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"

## 難易度 ★★☆☆☆☆☆☆

1. 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？
2. 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？
3. かけ算九九の表を出してみましょう。
4. 計算ゲームです。10 項 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。
5. 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。
6. 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？ 二番目に小さい数は何でしょうか？
7. 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。
8. 4680 秒は、何時間何分何秒でしょうか？
9. 干支を求めてみましょう。2000 年生まれの人の干支は何でしょうか？
10. 世界の人口は、73 億 2400 万人です。  
一年に 1.2 %ずつ人口が増えています。  
来年には何人になっているでしょうか？  
100 億人になる年はいつでしょうか？
11. 二の乗数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？
12. 消費税を求めてみましょう。  
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でしょうか？
13. 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：  
1等：世界一周の旅。  
2等：蒲郡温泉一泊二日。  
3等：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。

## 難易度 ★★★☆☆☆☆☆

1.  $1 + 2 + 3 + \dots$  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？
2. 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。

```

3. 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)
4. 円の面積を求めてみましょう。  
円周率は、3.14...と続きますが、よく使うので、C言語に用意されています。

```

#include <math.h> // 円周率 M_PI が定義されています
#include <stdio.h>
main(){
    printf("円周率 = %f", M_PI); // 特に書式を指定しなかった場合
    printf("円周率 = %.4f", M_PI); // 小数点以下4桁表示
    printf("円周率 = %.15f", M_PI); // 小数点以下15桁表示
}

```

5. 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。

```

6. 計算ゲームです。  
2桁にしたり、引き算や掛け算、割り算もできる様にしてみましょう。
7. 健康管理アプリです。  
朝食、昼食、夕食のカロリーの入力を求めてみましょう。  
"食べ過ぎです"  
"ちょうどです"  
と表示しましょう。

8. 太りすぎ, 痩せすぎの指標としてBMIがあります.

体重[単位kg]/(身長[単位m]の二乗)

と計算します.

身長と体重を入れると, 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)

9. 今年が平年か閏年かを求めるプログラムを作ってみましょう.

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.

10. 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.

例)302 -> 三百二

11. 成績管理システムを創ってみましょう.

10人の生徒は国語, 算数, 理科, 社会を学んでいます.

平均点, 最高点を求めてみましょう.

## 難易度 ★★★★☆☆☆☆

1. 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。
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。  
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？
3. 完全数とは次のような数のことです。  
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。  
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？
4. 英単語帳アプリを創ってみましょう。

```
I      -> わたし  
love  -> 愛する  
you   -> あなた
```

と答えられたら、満点です。  
10問出題して、英語力向上を目指しましょう。

5. 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。

## 難易度 ★★★★★☆☆

1. 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？
2. 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。（西暦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曜日が計算出来ます）

```

int y; // 西暦年
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;

```

3. 数当てゲーム(Match Number)  
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 925 だとします。  
999と入れると、9は正解ですので、1つ正解と表示します。  
520と入れると、5と2は正解ですので、2つ正解と表示します。  
592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、925を当てるゲームです。

4. 自動販売機を創ってみましょう。  
取り扱い商品は、次のとおりです。

プレミアムコーヒー	150円	3本.
コーヒー	120円	10本.
お茶	100円	20本.
スポーツドリンク	120円	10本.
エナジードリンク	200円	5本.

1000円札でプレミアムコーヒーを買った場合、  
500円玉1枚、100円玉3枚 50円玉1枚でおつりを出してみましょう。  
在庫管理機能や、売れ筋商品機能もつけてみましょう。

「777」になったのでもう一本買える機能もあると楽しいですね。

## 難易度 ★★★★★☆☆

1. オセロや将棋を創ってみましょう.

盤面を表示する機能.

手を入力する機能.

勝ち負けの判定機能もあるといいですね.

他にはどんな機能が必要でしょうか?

交互に, 人対人で対局しましょう.

コンピュータとの対戦もできるようにすると楽しいですね.

2. ブラックジャックや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, トランプの絵柄が表示されるともっと雰囲気が出ますね.

## 難易度 ★★★★★★★

1. 電卓を創ってみましょう.

(実行例)

計算式を入力して下さい>  $3 + 12 =$

15

計算式を入力して下さい>  $3 + 12 \cdot 1. 2 =$

27

計算式を入力して下さい>  $3 + 12 \cdot 1. (2 + 1) =$

39

計算式を入力して下さい>  $(3 + 12 \cdot 1. (2 + 1)) / 3.9 =$

10

計算式を入力して下さい>  $(3 + 12 \cdot 1. (2 + 1)) / 3.9 / 0 =$

エラーです。0での割り算は定義されていません。

一桁の数、二桁の数 正しく読み込み出来るでしょうか？

掛け算や割り算の優先順位は反映されていますか？

小数点の取り扱いは出来ているでしょうか？

() を使って優先順位を変えるにはどのようにしたら良いのでしょうか？

0で割られた場合にエラーメッセージの表示も必要ですね。

(ヒント)

逆ポーランド記法 スタック

逆ポーランド記法 二分木

で検索してみて下さい。

## C言語 練習課題 解答例

練習課題の解答例です。

いろいろな書き方で、課題の仕様を満たすプログラムを創ることが出来ます。

学習の参考にして下さい。

- 難易度 ★☆☆☆☆☆☆☆
- 難易度 ★★☆☆☆☆☆☆
- 難易度 ★★★☆☆☆☆☆
- 難易度 ★★★★☆☆☆☆
- 難易度 ★★★★★☆☆☆
- 難易度 ★★★★★★★☆
- おまけ

## 解答例 ★☆☆☆☆☆☆☆

1. 「おはようございます」と挨拶.

[goodmorning.c](#)

```
#include <stdio.h>

int main(int argc, char const *argv[]) {
    printf("おはようございます。\\n");
    return 0;
}
```

2. 「今、何時？」と聞いて挨拶.

[greeting.c](#)

```
#include <stdio.h>

int main(int argc, char const *argv[]) {

    // 変数の宣言
    int what_time_is_it_now; // 今何時か、格納するための変数

    printf("今何時ですか？\\n"); // 入力を促すために、メッセージを表示
    scanf("%d", &what_time_is_it_now); // 整数型の数字を、受け取る

    if (what_time_is_it_now < 12) { // 午前中
        printf("おはようございます。\\n"); // '\\n' は「改行文字」で、改行されます。
    } else if (what_time_is_it_now < 18) { // 夕方まで
        printf("こんにちは。\\n");
    } else { // 夜なら
        puts("おやすみなさい。");
    }
    return 0;
}
```

3. 計算プログラム.

[calculation.c](#)

```
#include <stdio.h>

int main(int argc, char const *argv[]) {

    // %d は整数型の書式指定子です。
    // 計算結果を直接表示指せることができます。
    printf("%d\n", 3 + 8 * 3);

    // 変数に代入してから、出力することも出来ます。
    int answer;           // 変数の宣言。整数型の変数 answer を宣言しています。
    answer = 18 / 3 - 1; // 変数への代入。
                           // 割り算は「÷」の代わりに「/」を使います。
    printf("%d\n", answer);

    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。
    printf("%d\n", result);

    return 0;
}
```

## 4.A4用紙の面積.

[a4\\_paper.c](#)

```
#include <stdio.h>

int main(int argc, char const *argv[]) {

    double height = 29.7;      // 小数を扱うには、double型として宣言します。
    double width = 21.0;

    double area;
    area = height * width;    // 掛け算は「×」の代わりに「*」を使います。

    printf("%f\n", area);    // 小数を表示する際は、%fを使います。

    return 0;
}
```

## 5.横棒グラフを表示.

[bar\\_graph1.c](#)

```
#include <stdio.h>

int main(int argc, char const *argv[]) {

    // 横棒グラフを表示します。
    // 直接、表示することも出来ます。
    printf("■■■■■\n");
    printf("■■■■□\n");
    printf("■■■□□\n");
    printf("■■□□□\n");
    printf("■□□□□\n");

    return 0;
}
```

[bar\\_graph2.c](#)

```
#include <stdio.h>

int main(int argc, char const *argv[]) {

    // 変数名は、i でも良いですが、
    // 行の変数であることが分かるようにすると良いです。
    int row;          // 行
    int black_box;   // 黒い■の数
    int white_box;   // 白い□の数

    for (row = 0; row < 5; row++) {
        black_box = 5 - row; // 黒い■の数
        white_box = row;    // 白い□の数
        // 初期値は設定済みなので、省略出来ます。
        for(; black_box > 0; black_box--){
            printf("■");    // ■を表示します。
        }
        for(; white_box > 0; white_box--){
            printf("□");    // □を表示します。
        }
        printf("\n");       // 改行します。
    }

    return 0;
}
```

bar\_graph3.c

```
#include <stdio.h>

// 自作の関数を創って、解くことも出来ます。

// 関数のプロトタイプ(原型)宣言
// 機能：黒い■と白い□を表示する
// 引数：int black_box // 黒い■の数
//         int white_box // 白い□の数
// 戻値：なし
void black_white_box(int black_box, int white_box);

int main(int argc, char const *argv[]) {
    // 変数名は、i でも良いですが、
    // 行の変数であることが分かるようにすると良いです。
    int row;           // 行
    int black_box;    // 黒い■の数
    int white_box;   // 白い□の数

    for (row = 0; row < 5; row++) {
        black_box = 5 - row; // 黒い■の数
        white_box = row;    // 白い□の数

        // 黒い■と白い□を表示する関数を呼び出し、
        // 表示をお任せします。
        black_white_box(black_box, white_box);
    }

    return 0;
}

// プロトタイプ宣言を先頭でしているので、
// 自作関数本体を、main関数の後に書くことが出来ます。
void black_white_box(int black_box, int white_box){
    // C言語では、while(0)が偽となり、while文が終了するので、
    // このようにも書くことが出来ます。
    while(black_box--){
        printf("■"); // ■を表示します。
    }
    while(white_box--){
        printf("□"); // □を表示します。
    }
    printf("\n"); // 改行します。
}
```

## 6.さいころを転がし、「偶数」「奇数」表示。

**dice.c**

```
#include <stdio.h>
#include <time.h> // time関数が定義されています。
#include <stdlib.h> // rand関数が定義されています。

int main(int argc, char const *argv[]) {

    int dice;

    // 実行した時刻によって、異なった乱数となるように、
    // 亂数の種を播きます。
    srand(time(NULL));

    // 6で割った余りに、1を加えることで、1～6までの乱数が得られます。
    dice = rand() % 6 + 1;

    printf("サイコロの目は、%d です。\\n", dice);

    // 2で割った余りが0なら、偶数、そうでないなら奇数です。
    if (dice % 2 == 0) {
        printf("偶数です\\n");
    } else {
        printf("奇数です\\n");
    }

    // 論理和を使って、このように書くことも出来ます。
    if (dice == 2 || dice == 4 || dice == 6){
        printf("偶数です\\n");
    } else {
        printf("奇数です\\n");
    }

    // switch case 文を使って書くことも出来ます。
    switch (dice) {
        case 1:
            printf("奇数です\\n");
            // このbreak文がないと、
            // case 2: も続けて実行されますので、
            // break文を書いています。
            break;
        case 2:
            printf("偶数です\\n");
    }
}
```

```

        break;
case 3:
    printf("奇数です\n");
    break;
case 4:
    printf("偶数です\n");
    break;
case 5:
    printf("奇数です\n");
    break;
case 6:
    printf("偶数です\n");
    break;
default:
    printf("エラーです\n");
    break; // default文のbreak;は不要ですが、対称性の観点から付けています。
}

// プログラムは、一行に一文が基本ですが、
// 簡単なswitch case 文であれば、
// 以下のように一行に書いても分かりやすいです。
switch (dice) {
    case 1:    printf("奇数です\n");    break;
    case 2:    printf("偶数です\n");    break;
    case 3:    printf("奇数です\n");    break;
    case 4:    printf("偶数です\n");    break;
    case 5:    printf("奇数です\n");    break;
    case 6:    printf("偶数です\n");    break;
    default:   printf("エラーです\n");    break;
}

// switch case 文を使って書くことも出来ます。
switch (dice) {
case 1:
    // 意図的に、break文を書かず、
    // case 5: まで、スルーさせて、
    // 奇数で在る旨、表示させています。
case 3:
case 5:
    printf("奇数です\n");
    break;
case 2:
case 4:
case 6:
    printf("偶数です\n");
}

```

```

        break;
    }

    return 0;
}

```

(自作関数化した例)

[dice2.c](#)

```

#include <stdio.h>
#include <time.h> // time関数が定義されています。
#include <stdlib.h> // rand関数が定義されています。

// 機能：0～max未満の乱数を返す
// 引数：int max 乱数の最大値
// 戻値：0～max未満の乱数
int random_number(int max){
    srand(time(NULL));
    return (rand() % max);
}

int main(int argc, char const *argv[]) {

    // 亂数はよく使うので、関数化してみました。
    // 同じプログラムを何度も書かずに済むので、便利です。
    // 関数のプロトタイプ宣言をしていないので、
    // mainの前に、random_number関数は書かれている必要があります。

    int dice = random_number(6) + 1;
    printf("サイコロの目は、%d です。\\n", dice);

    // 2で割った余りが0なら、偶数、そうでないなら奇数です。
    if (dice % 2 == 0) {
        printf("偶数です\\n");
    } else {
        printf("奇数です\\n");
    }

    return 0;
}

```

(外部ヘッダーファイルにした例)

[random\\_number.h](#)

```
// 自作のrandom_number.hです。
// random_number関数そのままです。

// 機能：0～max未満の乱数を返す
// 引数：int max 亂数の最大値
// 戻値：0～max未満の乱数
int random_number(int max){
    srand(time(NULL));
    return (rand() % max);
}
```

### dice3.c

```
#include <stdio.h>
#include <time.h> // time関数が定義されています。
#include <stdlib.h> // rand関数が定義されています。

// 自作したrandom_number()関数が定義されています。
// 自作のヘッダーファイルを読み込む際は、
// "(ダブルクォーテーション)で囲みます。
#include "random_number.h"

int main(int argc, char const *argv[]) {

    // random_number.hを読み込んでいるので、
    // すぐに、random_number関数を使えます。

    int dice = random_number(6) + 1;
    printf("サイコロの目は、%d です。\\n", dice);

    // 2で割った余りが0なら、偶数、そうでないなら奇数です。
    if (dice % 2 == 0) {
        printf("偶数です\\n");
    } else {
        printf("奇数です\\n");
    }

    return 0;
}
```

7.傘を持っていった方がよいか、アドバイス。

### umbrella.c

```
#include <stdio.h>
#include <time.h> // time関数が定義されています。
#include <stdlib.h> // rand関数が定義されています。

// 機能：0～max未満の乱数を返す
// 引数：int max 乱数の最大値
// 戻値：0～max未満の乱数
int random_number(int max);

// 0～max未満の乱数を返す関数
int random_number(int max){
    srand(time(NULL));
    return (rand() % max);
}

int main(int argc, char const *argv[]) {
    // 亂数はよく使うので、関数化してみました。
    // 同じプログラムを何度も書かずに済むので、便利です。

    // 降水確率 0 - 100 % の範囲の乱数
    // (補完機能が働くので、
    // 長い変数名でも差し支えありませんが、
    // さすがに長すぎるかもですね)
    int precipitation_probability = random_number(101);

    // printf文の書式の中で、「%」を表示させるには、「%%」と記述します。
    printf("降水確率は、%d %\n", precipitation_probability);

    // if else 文の条件式は、
    // 大きい順、あるいは、小さい順にして、
    // 順次、条件に合致させるようにしましょう。
    if (precipitation_probability <= 20){
        printf("傘はいらないです\n");
    } else if (precipitation_probability < 90){
        printf("持って行った方がいいかも\n");
    } else {
        printf("絶対持って行きましょう\n");
    }

    return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h> // atoi関数用

int main(int argc, char const *argv[]) {

    // コマンドラインから、降水確率を渡すことも出来ます。
    if (argc == 1) {
        printf("【使い方】\n");
        printf("傘を持っていくべきか、アドバイスするプログラムです。\n");
        printf("もし降水確率が、30% なら、\n");
        printf("%s 30\n", argv[0]); // argv[0] はプログラム自身の名前です。
        printf("と入力して下さい。.\n");

        exit(1); // プログラムを終了します。
    }

    // atoi関数は、文字列を数値に変換する関数です。
    // argv[1]が 文字列 "30" なら、
    // precipitation_probability には、整数 30 が入ります。
    int precipitation_probability = atoi(argv[1]);

    if (precipitation_probability <= 20){
        printf("傘はいらないです\n");
    } else if (precipitation_probability < 90){
        printf("持って行った方がいいかも\n");
    } else {
        printf("絶対持って行きましょう\n");
    }

    return 0;
}
```

## 解答例 ★★★☆☆☆☆☆☆

1.毎日 1 %ずつこつこつ成長.

one\_percent.c

```
#include <stdio.h>

int main(int argc, char const *argv[]) {

    double capability = 1.0; // 能力
    int i;

    for (i = 0; i < 365; i++){
        capability *= 1.01;
        // capability = capability * 1.01; と書くことも出来ます。
        // 自分自身に何かの計算を行い、
        // 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、
        // 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。
    }

    printf("一年後の能力は、\n");
    printf("%.15f\n", capability); // 小数点以下15桁表示
    printf("です。\n");
    printf("\n");
    printf("C言語 double型での計算結果\n");
    printf("37.783434332887275\n");
    printf("\n");
    printf("正確な値は、有効桁数732桁で、\n");
    printf("37.7834343328871588776166047964976054602711354915910020033039338
93694442952198593811935639436889138752947230257466652966950262937798745172
33301507922233862428614682541680615253144396919455694277651724794006295820
21756049578068333205496182837603299207844744407482328235228487747766633770
98517634258918092249275355047751709109700563151616706856329170679969143031
11984143610198730361066503225373596290071532034477267109474634224398074728
85377480448108054315136562847283771508607255440695157041803096694610715506
27216255083200959680558767329997739256425299018230968108183790782834451122
34139169902672871880967067586849494180180014804369532225491871467707211395
50421573105249454013216994798432008271425308713028897301180251050440194336
501\n");
    printf("です。\n");
    // 小数の計算は誤差がつきものです。
    // double型で計算しましたが、有効桁数 14 桁でした。

    return 0;
}
```

## 2. 「今、何時？」 と聞いて挨拶。

[fixed\\_deposit.c](#)

```
#include <stdio.h>

int main(int argc, char const *argv[]) {

    double fixed_deposit = 1.0; // 定期預金
    int     year = 0;           // 預けた年数

    // 繰り返す回数が分からない場合は、while文を使います。
    // 定期預金が2倍未満の間、繰り返します。
    while(fixed_deposit < 2.0) {
        fixed_deposit = fixed_deposit * 1.03;
        // fixed_deposit *= 1.03; // と書くことも出来ます。
        year++;
    }

    printf("今年預けた定期預金は %d 年後に %5.3f 倍になりました。\n",
           // 長くなった場合、カンマの後で改行することも出来ます。
           year, fixed_deposit);

    return 0;
}
```

## 3.かけ算九九の表.

[multiplication\\_table.c](#)

```
#include <stdio.h>

int main(int argc, char const *argv[]) {

    int multiplier; // 乗数 (掛ける数)
    int multiplicand; // 被乗数 (掛けられる数)

    // 掛け算九九の表を表示する
    printf(" 一 二 三 四 五 六 七 八 九\n");
    printf(" の の の の の の の\n");
    printf(" 段 段 段 段 段 段 段 段\n");
    for (multiplier = 1; multiplier <= 9; multiplier++){
        for (multiplicand = 1; multiplicand <= 9; multiplicand++){
            printf("%3d", multiplicand * multiplier);
        }
        printf("\n");
    }

    return 0;
}
```

#### 4.計算ゲーム.

##### [util.h](#)

```
#include <time.h> // time関数が定義されています。
#include <stdlib.h> // rand関数が定義されています。

// 機能：0～max未満の乱数を返す
// 引数：int max 乱数の最大値
// 戻値：0～max未満の乱数
int random_number(int max);

// 0～max未満の乱数を返す関数
int random_number(int max){
    srand(time(NULL));
    return (rand() % max);
}
```

##### [calculation\\_game1.c](#)

```
#include <stdio.h>
#include "util.h" // 自作関数いろいろの定義
```

```

int main(int argc, char const *argv[]) {

    int operand1;          // 第一被演算子
    int operand2;          // 第二被演算子
    int operation_result; // 演算結果
    int your_answer;       // 回答
    int correct_answer;    // 正答数
    int i;

    correct_answer = 0; // 10問中何問正解か数えるために、初期化します。
    // 1回目、2回目と表示させたいので、
    // for (i = 1; i <= 10; i++) と書いているところに着目して下さい。
    for (i = 1; i <= 10; i++){
        // 出題処理
        operand1 = random_number(10);
        // 二つの0～9までの数を用意します。
        // do while 文で、異なる数になるまで、繰り返します。
        do {
            operand2 = random_number(10);
        } while(operand2 == operand1);

        operation_result = operand1 + operand2;      // 演算結果
        printf("足し算ゲーム %d 回目", i);
        printf("%d + %d = ?\n", operand1, operand2);

        // 回答を受け取る
        scanf("%d", &your_answer);
        while(getchar() != '\n'); // キーバッファ読み飛ばす

        // 正解発表と正答数のカウント
        if (your_answer == operation_result){
            printf("正解です。\\n");
            correct_answer++;
        } else {
            printf("正解は、%d です。\\n", operation_result);
        }
    }

    // 総合結果発表
    printf("10問中 %d 問 正解です。\\n", correct_answer);

    return 0;
}

```

## 5.配列の10個の数字の合計。

**summation.c**

```
#include <stdio.h>
#include "util.h" // 自作関数いろいろの定義

int main(int argc, char const *argv[]) {

    // 10個の数字が入った配列 繼めて初期値を与えています。
    // 配列は、array (アレイ) と言います。
    int array[] = { 3, 15, 22, 81, 41, 0, 83, 72, 50, 33 };

    int sum; // 合計
    int i;

    // 合計処理
    sum = 0; // 初期化します。
    for (i = 0; i < 10; i++){
        sum = sum + array[i];
        // sum += array[i]; と書くことも出来ます。
    }

    // 結果発表
    for (i = 0; i < 10; i++){
        printf("array[%d]: %2d \n", i, array[i]);
        // sum += array[i]; と書くことも出来ます。
    }
    printf(" 合計は %d です。 \n", sum);

    return 0;
}
```

6.配列の10個の数字一番小さい数二番目に小さい数.

**minimum\_number.c**

```
#include <stdio.h>
#include <limits.h> // 整数の取り得る範囲が定義されています。
#include "util.h" // 自作関数いろいろの定義

int main(int argc, char const *argv[]) {

    // 10個の数字が入った配列 繼めて初期値を与えています。
    // 配列は、array (アレイ) と言います。
    // (簡単化のために、同じ数字はないものと仮定しています。)
    int array[] = { 3, 15, 22, 81, 41, 83, 72, 0, 50, 33 };
```

```

int min;           // 最小値を格納する変数
min = INT_MAX;   // min = 999; と書いてもいいのですが、
                  // ここでは、整数型の取り得る最大値で初期化しています。
int i;

// 最小値を求める処理
// 最初、min に一番大きい数を入れておきます。
// そして、配列内のそれぞれの要素と比較することで、
// 一番、小さい数が、minにセットされます。
for (i = 0; i < 10; i++){
    if (min > array[i]) {
        min = array[i];
    }
}
// 結果発表
for (i = 0; i < 10; i++){
    printf("array[%d]: %2d \n", i, array[i]);
}
printf("一番小さい数は、%d です。 \n", min);

// 二番目に小さい数を求めます。
// int array[] = { 3, 15, 22, 81, 41, 83, 72, 0, 50, 33 };
// から、一番小さい数 0 を除いた
// int array[] = { 3, 15, 22, 81, 41, 83, 72, 50, 33 };
// の中から、一番小さい数を調べたら、良いことになります。
// 0を取り除くには、どうしたらよいのでしょうか？
// 取り除くのではなく、後ろの要素を前に詰める！ と考えます。
// 0 は 7 番目の要素ですから、
// array[7] に array[8] を代入して、
// array[8] に array[9] を代入すれば、詰めたことになります。
// そうすると、
// int array[] = { 3, 15, 22, 81, 41, 83, 72, 50, 33, 33 };
// という配列になります。

// それでは、0 が array の何番目の要素であったかを求めてみましょう。
int index = 0; // 最小値 min が何番目の要素であるか
for (i = 0; i < 10; i++){
    if (array[i] == min){
        index = i;
    }
}

// index には、7番目と入っています。
// 後ろの要素を前の要素に詰めます。

```

```

// array[7] に array[8] を代入して、
// array[8] に array[9] を代入します。
for (i = index; i < 9; i++){
    array[i] = array[i + 1];
}

// 詰まっているかどうか、確認の為に出力してみましょう。
printf("詰めた結果\n");
for (i = 0; i < 10; i++){
    printf("array[%d]: %2d \n", i, array[i]);
}

// 二番目に小さい数を調べます。
min = INT_MAX;
for (i = 0; i < 9; i++){ // 10まで調べなくても、9までOKです。
    if (min > array[i]) {
        min = array[i];
    }
}
// 結果発表
printf("二番目に小さい数は、%d です。 \n", min);

return 0;
}

```

7.配列の10個の数字 大きい順に並び替え.

#### [selection\\_sort.c](#)

```

#include <stdio.h>
#include <limits.h> // 整数の取り得る範囲が定義されています。
#include "util.h" // 自作関数いろいろの定義

int main(int argc, char const *argv[]) {

    // 10個の数字が入った並び替え前の配列
    int array[] = { 3, 15, 22, 81, 41, 83, 72, 0, 50, 33 };

    int min; // 最小値を格納する変数
    int i;

    // 最小値を求める処理
    min = INT_MAX;
    for (i = 0; i < 10; i++){ // (a)
        if (min > array[i]) {

```

```
    min = array[i];
}
}

// それでは、0 が array の何番目の要素であったかを求めてみましょう。
int index = 0; // 最小値 min が何番目の要素であるか
for (i = 0; i < 10; i++){ // (b)
    if (array[i] == min){
        index = i;
    }
}

printf("(a)(b)それぞれ行った場合の結果表示\n");
for (i = 0; i < 10; i++){
    printf("array[%d]: %2d \n", i, array[i]);
}
printf("min: %d index: %d\n", min, index);

// ここで、(a) と (b) 同時に出来るのではと思えて来ます。
// 一緒に書くと、次のようにになります。
// 最小値を求めると同時に、最小値 min が何番目の要素であるか求める
min = INT_MAX;
for (i = 0; i < 10; i++){
    if (min > array[i]) {
        min = array[i]; // (a)
        index = i; // (b)
    }
}

printf("(a)(b)纏めて行った場合の結果表示\n");
for (i = 0; i < 10; i++){
    printf("array[%d]: %2d \n", i, array[i]);
}
printf("min: %d index: %d\n", min, index);

// index には、7番目と入っています。
// 後ろの要素を前の要素に詰めます。
// array[7] に array[8] を代入して、
// array[8] に array[9] を代入します。
for (i = index; i < 9; i++){
    array[i] = array[i + 1];
}

// 前回は、array[9] は特に処理せず、そのままでしたが、
// 今回は、array[9] に、今求めた、一番小さい数を入れます。
```

```
array[9] = min;

// 一番小さい数が、array の最後に来ているはずです。
// 確認の為に出力してみましょう。
printf("一番小さい数が、最後に来ていることの確認\n");
for (i = 0; i < 10; i++){
    printf("array[%d]: %2d \n", i, array[i]);
}

// これで、一番小さい数を、最後に持つて行くことが出来ました。
// 次は、0番目～8番目で並び替え、
// その次は、0番目～7番目で並び替え、
// 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。

// 完成版のプログラムです。
// 0番目～何番目までを並び替えるのか、
// 管理するためのfor文で外側をくるんでいます。
// この並び替えのアルゴリズムを、選択ソート と言います。

int sorted; // 今、何番目の要素まで並び替えが終了しているのか。

for (sorted = 9; sorted >= 1; sorted--){
    min = INT_MAX;
    for (i = 0; i <= sorted; i++){
        if (min > array[i]) {
            min = array[i]; // (a)
            index = i; // (b)
        }
    }
    for (i = index; i < sorted; i++){
        array[i] = array[i + 1];
    }
    array[sorted] = min;
}

printf("\n");
printf("【選択ソート】並び替え結果\n");
for (i = 0; i < 10; i++){
    printf("array[%d]: %2d \n", i, array[i]);
}

printf("\n");
printf("選択ソートの他に、有名な並び替え方法としては、\n");
printf("クイックソート\n");
printf("バブルソート\n");
```

```

printf("マージソート\n");
printf("が、あるので、学習してみて下さい。\\n");

return 0;
}

```

8.4 6 8 0秒は、何時間何分何秒。

#### time.c

```

#include <stdio.h>
#include <time.h> // 時刻に関する関数がいろいろ用意されています。（今回未使用）

int main(int argc, char const *argv[]) {

    int remain_time = 4680;

    int hour;
    int minute;
    int second;

    hour      = remain_time / 3600; // 3600秒で割った商が時間です。
    remain_time = remain_time % 3600; // 3600秒で割った余りが残り時間です。
    minute    = remain_time / 60;   // 60秒で割った商が分です。
    remain_time = remain_time % 60; // 60秒で割った余りが残り時間です。
    second    = remain_time;

    printf("4680秒は、%d 時間 %d 分 %d 秒 です。\\n", hour, minute, second);

    // 時刻に関する関数を使って求めるこども出来ます。
    // 興味のある方は、挑戦されて下さい。

    return 0;
}

```

9.干支を求める。

#### eto.c

```

#include <stdio.h>

int main(int argc, char const *argv[]) {

    int year = 2000;

```

```
// switch case 文で、  
// 余りに応じて、それぞれの干支を表示します。  
switch (year % 12) {  
    case 0:  
        printf("申年\n");  
        break;  
    case 1:  
        printf("酉年\n");  
        break;  
    case 2:  
        printf("戌年\n");  
        break;  
    case 3:  
        printf("亥年\n");  
        break;  
    case 4:  
        printf("子年\n");  
        break;  
    case 5:  
        printf("丑年\n");  
        break;  
    case 6:  
        printf("寅年\n");  
        break;  
    case 7:  
        printf("卯年\n");  
        break;  
    case 8:  
        printf("辰年\n");  
        break;  
    case 9:  
        printf("巳年\n");  
        break;  
    case 10:  
        printf("午年\n");  
        break;  
    case 11:  
        printf("未年\n");  
        break;  
}  
  
// 一行に一文が基本ですが、  
// この場合は、以下のように複数行書いても  
// 見やすいでしょう。
```

```

switch (year % 12) {
    case 0: printf("申\n"); break;
    case 1: printf("酉\n"); break;
    case 2: printf("戌\n"); break;
    case 3: printf("亥\n"); break;
    case 4: printf("子\n"); break;
    case 5: printf("丑\n"); break;
    case 6: printf("寅\n"); break;
    case 7: printf("卯\n"); break;
    case 8: printf("辰\n"); break;
    case 9: printf("巳\n"); break;
    case 10: printf("午\n"); break;
    case 11: printf("未\n"); break;
}

// 配列のそれぞれの要素に、干支を格納すると、
// もっとシンプルに書くことが出来ます。

// 12で割った余りが、配列の添え字になるようにしているのがポイントです。
// "申"と一文字に見えますが、
// 文字コードが
// utf-8 の場合、3文字(バイト) E7 94 B3 です。(shift-jis の場合、2文字(バイト)
90 5C)
// ですので、'E7', '94', 'B3', '\n' の4文字の配列で、"申" を表せます。
// char saru[4] = { 'E7', '94', 'B3', '\n' };
// で、干支は12要素ありますから、
// 配列の配列(=二次元配列)にすれば、干支の配列になります。
char eto[][4] = { "申", "酉", "戌", "亥", "子", "丑", "寅", "卯", "辰", "巳",
", "午", "未" };
// 配列の一次元目の[4]は必要です。
// 4を書かずに、[]とだけ書かれると、コンパイルエラーとなります。
// ポインタを使うと、
// char *eto[] = { "申", "酉", "戌", "亥", "子", "丑", "寅", "卯", "辰", "巳",
", "午", "未" };
// と書くことも出来ます。

int index;
index = year % 12;
printf("あなたの干支は、%s です。\\n", eto[index]);

// 一行に纏めて書くことも出来ます。
printf("あなたの干支は、%s です。\\n", eto[year%12]);

return 0;
}

```

## 10.世界の人口.

## population.c

```
#include <stdio.h>
#include <string.h>

int main(int argc, char const *argv[]) {

    long population = 7324000000; // 世界人口
                                // 約21億以上の数は、long型を使います。
    double growth_rate = 0.012; // 人口増加率 1.2%

    // int, double, long 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。
    long next_year_population
        = population * (1 + growth_rate);

    // long型の書式指定子は、%ld です。
    printf("来年の人口は %ld 人です。\\n", next_year_population);

    // 何年後に100億人になるか、求めてみましょう。
    // 繰返し回数が不明な場合は、while文を使います。
    int year = 0;
    while (population < 1e10){ // 1e10 は、10の10乗(=100億)です。
        population *= (1 + growth_rate);
        year++;
        printf("%d 年後の人口は %ld 人です。\\n", year, population);
    }

    // おまけ

    // 3桁ごとにカンマ区切りで出してみましょう。
    // 数としての人口を文字列に変換します。
    char str_population[32]; // 世界人口が入った文字列
    char str_population_reverse[32]; // 逆順になっている文字列（作業用）
    char str_comma_population_reverse[32]; // カンマを入れて逆順になっている文字
    列（作業用）
    char str_comma_population[32]; // カンマを入れた文字列
    sprintf(str_population, "%ld", next_year_population);
    int digit = strlen(str_population);
    printf("%d 桁の数です。\\n", digit);

    //
    // str_population      : 7411888000
    // str_comma_population : 7,411,888,000
}
```

```

// カンマは、最後の桁から3つごとに入れていきます。
// 最後から数えて3つずつカンマを入れていくのは、
// 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、
// 逆順に並び替えます。
// 7411888000 -> 0008881147
int i;
for (i = 0; i < digit; i++){
    str_population_reverse[digit - i - 1] = str_population[i];
}
str_population_reverse[digit] = '\0'; // 文字列の終わりが分かるように最後に'\0'
'を入れます。
printf("逆順に並び替えて、%s となりました。\\n", str_population_reverse);

// 3桁ごとにカンマを入れていきます。
int comma = 0; // 今までに入れたカンマの数
for (i = 0; i < digit; i++){
    str_comma_population_reverse[i + comma] = str_population_reverse[i];
    // 0桁目、1桁目では何もしませんが、2桁目のコピーが終わった後に、
    // カンマを追加する処理を行います。
    if (i % 3 == 2) {
        comma++;
        str_comma_population_reverse[i + comma] = ',';
    }
}
str_comma_population_reverse[digit + comma] = '\0'; // 文字列の終わりが分かるように最後に'\0'を入れます。
printf("カンマを追加して、%s となりました。\\n", str_comma_population_reverse);

// もう一度、並び替えます。
for (i = 0; i < digit + comma; i++){
    str_comma_population[digit + comma - i - 1] = str_comma_population_rev
erse[i];
}
str_comma_population[digit + comma] = '\0'; // 文字列の終わりが分かるように最後に'\0'を入れます。
printf("もう一度並び替えて、%s となりました。\\n", str_comma_population);

return 0;
}

```

## 11.二の乗数.

**power.c**

```
#include <stdio.h>
#include <math.h> // pow 関数

int main(int argc, char const *argv[]) {

    // 2の幂乗(べきじょう)を求めます。

    int n;
    long power = 1;
    for (n = 1; n <= 20; n++){
        power *= 2;
        printf("2 の %2d 乗は %8ld です。\\n", n, power);
    }

    // 幂乗を求める関数も用意されています。
    for (n = 1; n <= 20; n++){
        power = pow(2, n);
        printf("2 の %2d 乗は %8ld です。\\n", n, power);
    }

    // 16進数で出力する際は、%x を指定します。
    // ここでは、power が long 型なので %lx を指定します。
    // 10進数と16進数との対応も参考までに出力してみます。
    printf("乗数 16進数          10進数\\n");
    for (n = 1; n <= 16; n++){
        power = pow(2, n);
        printf("%2d %11lx %13ld\\n", n, power, power);
    }
    for (n = 20; n <= 40; n+=10){
        power = pow(2, n);
        printf("%2d %11lx %13ld\\n", n, power, power);
    }

    return 0;
}
```

乗数	16進数	10進数
1	2	2
2	4	4
3	8	8
4	10	16
5	20	32
6	40	64
7	80	128
8	100	256
9	200	512
10	400	1024
11	800	2048
12	1000	4096
13	2000	8192
14	4000	16384
15	8000	32768
16	10000	65536
20	100000	1048576
30	40000000	1073741824
40	10000000000	1099511627776

2の10乗は、1k(キロ) 約1000

2の20乗は、1M(メガ) 約100万

2の30乗は、1G(ギガ) 約10億

2の40乗は、1T(テラ) 約1兆

## 12.消費税.

[consumption\\_tax.c](#)

```
#include <stdio.h>
#include <math.h> // floor, ceil関数

int main(int argc, char const *argv[]) {

    // 消費税を求めます。
    int ice_cream = 98;           // アイスクリームの値段
    double consumption_tax_rate = 0.08; // 消費税率
                                         // 変数にしておくと、変更があった場合の修正が楽です。
    double consumption_tax;        // 消費税額

    consumption_tax = ice_cream * 3 * consumption_tax_rate;

    printf("お買い上げ額は、      %-5d 円です。\\n", ice_cream * 3);           // %-5d で左詰で5桁表示されます。
    printf("消費税は、          %6.2f 円です。\\n", consumption_tax);
    printf("端数を切り捨てる、 %6.2f 円です。\\n", floor(consumption_tax)); // 切り捨て floorは床の意味です。
    printf("端数を切り上げる、 %6.2f 円です。\\n", ceil(consumption_tax)); // 切り上げ ceilは天井の意味です。

    return 0;
}
```

## 13.福引き.

[lottery.c](#)

```
#include <stdio.h>
#include <string.h>
#include "util.h" // 自作のrandom_number関数

int main(int argc, char const *argv[]) {

    // 福引きの等級名
    // 配列の添字は0から始まるので、先頭に""を入れています。
    // prize_rank[1] が "一等賞"だと分かりやすいですね。

    char *prize_rank[] = { "", "一等賞", "二等賞", "三等賞", "残念賞" };
    // 福引きの賞品
    char *prize_item[] = { "", "世界一周の旅", "蒲郡温泉一泊二日", "お好み焼き食べ放題", "ティッシュペーパー" };
    // メッセージ
```

```
char message[64];

// 亂数はよく使うので、流用しています。
int lottery = random_number(10) + 1; // 福引き 1-10の乱数が得られます。

int rank; // 何等賞か？
if (lottery == 1){
    rank = 1; // 一等賞
} else if (lottery >= 2 && lottery <= 3) {
    // と書いても同じですが、
} else if (2 <= lottery && lottery <= 3) {
    // と書いた方が、範囲が分かりやすくなります。
    rank = 2; // 二等賞
} else if (4 <= lottery && lottery <= 6) {
    rank = 3; // 三等賞
} else {
    rank = 4; // 残念賞
}

// 小さい順に切り取っているので、
// else if の下限は省略出来ます。
if (lottery == 1){
    rank = 1; // 一等賞
} else if (lottery <= 3) {
    rank = 2; // 二等賞
} else if (lottery <= 6) {
    rank = 3; // 三等賞
} else {
    rank = 4; // 残念賞
}

// ちなみに条件式を書けないので、switch case 文には出来ないです。

// それぞれの等級に応じたメッセージを創っています。
// 無理矢理、文字列操作の関数を使っています。
strcpy(message, "おめでとう！");      // コピー関数
strcpy(message, prize_rank[rank]);    // コピー関数
strcat(message, ":");                // 連結関数
strcat(message, prize_item[rank]);   // 連結関数
strcat(message, " が当たったよ。"); // 連結関数
printf("%s\n", message);

return 0;
}
```

## 解答例 ★★★★☆☆☆☆☆

1.1+2+3...

total\_10000.c

```
#include <stdio.h>

int main(int argc, char const *argv[]) {

    int n;
    int total;

    // まず足してから10000を越えるか、判定したいので、
    // do while 文を使いました。
    // 後判定ループとも言います。
    // 条件式が、total <= 10000 と
    // 10000以下の中は繰り返すようになっている点も、着目して下さい。
    n      = 1;
    total = 0;
    do {
        total = total + n;
        n++;
        // 一行に纏めて書くことも出来ます。
        // total += n++;
    } while (total <= 10000);
    printf("%d を足したら、10000を越えて %d になりました。\\n", --n, total);

    // while 文を使って書くことも出来ます。
    // 合計を求める処理が重複していることに着目して下さい。
    n      = 1;
    total = 0;
    total = total + n;
    n++;
    while (total <= 10000){
        total = total + n;
        n++;
    }
    printf("%d を足したら、10000を越えて %d になりました。\\n", --n, total);
```

```
// 重複を避けるために、次のように書くことも出来ます。
// 条件式が逆転していることに着目して下さい。
n      = 1;
total = 0;
while (1) {
    total = total + n;
    n++;
    if (total > 10000){
        break;
    }
}
printf("%d を足したら、10000を越えて %d になりました。\\n", --n, total);

return 0;
}
```

## 2. 縦棒グラフ.

[vertical\\_bar\\_graph.c](#)

```
#include <stdio.h>
#include <string.h>

int main(int argc, char const *argv[]) {

    // 縦棒グラフを表示する
    // y
    // ^
    // | □■□□
    // | □■■□□
    // | □■■■□
    // | □■■■■
    // | ■■■■■
    // -----> x

    // 表示させたい棒グラフの値
    int value[] = { 1, 4, 5, 3, 2 };
    // グラフ用の二次元配列
    char graph[5][5][4]; // [4]は、"□", "■"が、4文字必要なため。
    // char graph[5][5]; // '*', 'o' の一文字で表示するなら、これで良い。

    // graph[0]
    // □ graph[0][4]
    // □ graph[0][3]
    // □ graph[0][2]
```

```

// □ graph[0][1]
// ■ graph[0][0]

// value の値に応じて、棒グラフの長さをセットする
int x, y;
for (x = 0; x < 5; x++){
    int v = value[x];
    for (y = 0; y < 5; y++){
        if (y < v) {
            // graph[x][y] = "■"; // こう書きたいけれど、書けないので
            // graph[x][y] = '*'; // char 一文字なら、こう書ける
            strcpy(graph[x][y], "■");
        } else {
            strcpy(graph[x][y], "□");
            // graph[x][y] = 'o';
        }
    }
}

// グラフ表示
for (y = 4; y >= 0; y--){
    for (x = 0; x < 5; x++){
        printf("%s", graph[x][y]);
        // printf("%c", graph[x][y]); // '*', 'o' の一文字で表示するなら、これで良い。
    }
    printf("\n");
}

return 0;
}

```

## 3.昭和30年生まれの方の干支.

## eto2.c

```

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main(int argc, char const *argv[]) {

    // 入力を促すメッセージ表示
    printf("何年生まれですか? (例: S30) >\n");

```

```

// 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
char buffer[8];

// キーボードから一行読み込む
// scanfはいろいろ支障があるので、
// fgets関数で、bufferへ、sizeof(buffer)-1文字分(7文字分), stdin(=キーボード)
// から読み込む
if (fgets(buffer, sizeof(buffer), stdin) == NULL){
    // エラーメッセージ出力
    printf("キーボードから読み込めませんでした。\\n");
    exit(1);
}

// 改行文字が含まれているかどうか？
if ( strchr(buffer, '\\n') != NULL ) {
    // S30エンター のように4文字タイプされたときは、
    // 改行文字（エンター）を文字列の終端記号に置換する
    buffer[strlen(buffer)-1] = '\\0';
} else {
    // 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)
    // 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
    while(getchar() != '\\n');
}

// 読み込んでいるか、確認
// printf("キーボードからの入力は、\\n%s です。\\n", buffer);

// 入力文字列の先頭が、M, T, S, H で始まっているか確認し、西暦年に変換する
int year;           // 西暦年
char era_initial; // 明治: 'M', 大正: 'T', 昭和: 'S', 平成: 'H'
char era_year[3]; // 元号で何年か

// S30 と入力されていた場合、
// S をera_initialにセットする
era_initial = buffer[0];
// 30 をera_yearにセットする
era_year[0] = buffer[1];
era_year[1] = buffer[2];
era_year[2] = '\\0';

char *endptr; // 数値に変換出来なかった文字
// 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。
// 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、
// 区別が出来ないので、strtol関数を使います。

```

```
year = strtol(era_year, &endptr, 10); // 10進数として変換する

// 数値変換不可の文字列の長さを調べる。
if (strlen(endptr) != 0){
    // エラーメッセージ出力
    printf("%s は、元号年として認識出来ませんでした。\\n", endptr);
    exit(1);
}

switch (era_initial) {
case 'M':    year += 1867;    break;
case 'T':    year += 1911;    break;
case 'S':    year += 1925;    break;
case 'H':    year += 1988;    break;
default:
    // エラーメッセージ出力
    printf("%c は、元号の文字として認識出来ませんでした。\\n", era_initial);
    exit(1);
}

// 干支の算出方法は、前回と同様
// 干支の配列（12で割り切れる年は申年）
char *eto[] = { "申", "酉", "戌", "亥", "子", "丑", "寅", "卯", "辰", "巳",
"午", "未" };
printf("%c%s年(%d年)生まれのあなたの干支は、%s です。\\n",
       era_initial, era_year, year,
       eto[year % 12]);

return 0;
}
```

## 4.円の面積.

circle\_area.c

```
#include <stdio.h>
#include <math.h> // 円周率 M_PI が定義されています

int main(int argc, char const *argv[]) {

    int     radius = 10;                      // 円の半径
    double area   = pow(radius, 2) * M_PI;    // 円の面積
                                                // pow関数で、2乗を求めています。

    printf("円周率 = %f\n",      M_PI); // 特に書式を指定しなかった場合
    printf("円周率 = %.4f\n",     M_PI); // 小数点以下4桁表示
    printf("円周率 = %.15f\n",    M_PI); // 小数点以下15桁表示

    printf("半径 %d の円の面積は、%.15f です。\\n", radius, area);

    return 0;
}
```

## 5.英単語の長さ.

### [word\\_length.c](#)

```
*****
*
*
* 入力された英単語の長さを表示する
* (http://nzlife.net/archives/9581 に長い英単語の豆知識があります)
*
*****
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// #define で、TRUE という定数を1であると定義しています
// 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、
// 意味が分かりやすくて、よいです。
#define TRUE 1

int main(int argc, char const *argv[]) {
    char buffer[64]; // 英単語の読み込み用

    while(TRUE){
```

```

// 入力を促すメッセージの表示
printf("英単語を入力して下さい。(終了:bye)\n");

// ちなみにプログラム学習用なので、
// scanf("%s", buffer);
// の一行が簡単だったりします。

// キーボードから一行読み込む
// fgets関数で、bufferへ、sizeof(buffer)-1文字分(7文字分), stdin(=キーボード)
)から読み込む
if (fgets(buffer, sizeof(buffer), stdin) == NULL){
    // エラーメッセージ出力
    printf("キーボードから読み込めませんでした。\\n");
    exit(1);
}

// 改行文字が含まれているかどうか？
if (strchr(buffer, '\\n') != NULL) {
    // S30エンター のように4文字タイプされたときは、
    // 改行文字（エンター）を文字列の終端記号に置換する
    buffer[strlen(buffer)-1] = '\\0';
} else {
    // 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)
    // 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
    while(getchar() != '\\n');
}

// 無限ループとなっているので、
// 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 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する場合) 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 ((strcmp(buffer, "bye"))){
// と書いても同じですが、!= 0 と明示されていると、
// 分かりやすいかと思います。
if ((strcmp(buffer, "bye")) != 0){
    printf("入力された英単語は、%s ですね。\\n", buffer);
    printf("長さは、%lu 文字の単語ですね。\\n", strlen(buffer));
    printf("意味は・・・ う～ん分かりません。\\n");
    printf("\\n"); // 前の行に\\nを入れてもOKですが、
                // 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。
} else {
    printf("また、使ってね。bye-bye\\n");
    break;
}

```

```

    }
}

return 0;
}

```

## 6.計算ゲーム.

[calculation\\_game2.c](#)

```

*****
*
*
* 足し算ゲームとほぼ同様です。
* 術数と、演算の種類をリクエストするようになっています。
* 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。
*
*****
*/

```

```

#include <stdio.h>
#include <math.h>
#include <string.h>
#include "util.h" // 自作の乱数

int main(int argc, char const *argv[]) {

    int operand1;           // 第一被演算子
    int operand2;           // 第二被演算子
    char operator;          // 演算子 ('+', '-', '*', '/')
    int digit;              // 術数
    int operation_result;   // 演算結果
    int your_answer;        // 回答
    int correct_answer;     // 正答数
    int i;

    // 術数をリクエスト
    do {
        // scanf簡単です。
        // 文字を入力されたときは、無視します。
        printf("何桁の数字で挑戦しますか？\n");
        printf("一桁: 1    二桁: 2    三桁: 3    四桁: 4\n");
        scanf("%d", &digit);
        while(getchar() != '\n'); // キーバッファ読み飛ばす
    }
}
```

```

} while (digit <= 0 || digit >= 5);

// 演算子をリクエスト
do {
    printf("どの計算に挑戦しますか？\n");
    // 足し算、引き算、掛け算、割り算のことです。
    // 技術者なら、
    // 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。
    printf("加算: '+' 減算: '-' 乗算: '*' 除算: '/'\n");
    scanf("%c", &operator);
    while(getchar() != '\n'); // キーバッファ読み飛ばす
    // char型一文字なので、簡単に比較出来ます。
    // 全角で"+"や"-"なら文字列操作関数を使って下さい。
} while (operator != '+' && operator != '-' && operator != '*' && operator != '/');

correct_answer = 0; // 10問中何問正解か数えるために、初期化します。
// 1回目、2回目と表示させたいので、
// for (i = 1; i <= 10; i++) と書いているところに着目して下さい。
for (i = 1; i <= 10; i++){
    // 出題処理
    // 幕乗を求める関数を使って、必要な桁数に調整します。
    operand1 = random_number(pow(10, digit));
    // do while文で、異なる数になるまで、繰り返します。
    do {
        operand2 = random_number(pow(10, digit));
        // 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。
    } while(operand2 == operand1 || (operator == '/' && operand2 == 0));

    switch(operator){
        // switch case文には、文字型も使えます。
        case '+': operation_result = operand1 + operand2; break;
        case '-': operation_result = operand1 - operand2; break;
        case '*': operation_result = operand1 * operand2; break;
        case '/': operation_result = operand1 / operand2; break;
    }

    // ゲーム名を表示するための場合分けです。
    // こう書いた方が素直です。
    // switch(operator){
    // case '+':
    //     printf("足し算ゲーム %d回目", i);
    //     break;
    // case '-':
    //     printf("引き算ゲーム %d回目", i);
}

```

```

// break;
// case '*':
// printf("掛け算ゲーム %d 回目", i);
// break;
// case '/':
// printf("割り算ゲーム %d 回目", i);
// break;
// }

// 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
// 連想配列（ハッシュ）のご紹介
//
// ASCIIコード表を見ると、
// '+' 43
// '-' 45
// '*' 42
// '/' 47
// となっていますので、
// 47個+1個の大きさの文字列の配列を用意します。
char game_name[47+1][20];
strcpy(game_name['+'], "足し算");
// 書いて、配列の初期値を設定します。
// strcpy(game_name[43], "足し算");
// と書いたのと同じです。
strcpy(game_name['-'], "引き算");
strcpy(game_name['*'], "掛け算");
strcpy(game_name['/'], "割り算");
// 実際に printf("%s", game_name['+']);と書くと「足し算」と表示されます。
// このことをわきまえると、
printf("%sゲーム %d 回目\n", game_name[operator], i);
// と書けます。
//
// 配列は、添字として、数字をとりましたが、
// 文字を添字として使えるようになると、分かりやすかったり、
// 便利だったりします。
// game_name["+"] = "足し算";
// の様なイメージです。
// 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。
// 他言語学習の際の参考になさって下さい。

printf("%d %c %d = ?\n", operand1, operator, operand2);

// 回答を受け取る
scanf("%d", &your_answer);
while(getchar() != '\n'); // キーバッファ読み飛ばす

```

```

// 正解発表と正答数のカウント
if (your_answer == operation_result){
    printf("正解です。\\n");
    correct_answer++;
} else {
    printf("正解は、%d です。\\n", operation_result);
}
}

// 総合結果発表
printf("10問中 %d 問 正解\\n", correct_answer);

return 0;
}

```

(より均一な乱数が出るように改良)

#### mt.h

```

/*
A C-program for MT19937, with initialization improved 2002/1/26.
Coded by Takuji Nishimura and Makoto Matsumoto.

Before using, initialize the state by using init_genrand(seed)
or init_by_array(init_key, key_length).

Copyright (C) 1997 - 2002, Makoto Matsumoto and Takuji Nishimura,
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.

3. The names of its contributors may not be used to endorse or promote
   e

```

products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR

CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Any feedback is very welcome.

<http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html>  
email: m-mat @ math.sci.hiroshima-u.ac.jp (remove space)

\*/

/\*

The original version of <http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/MT2002/CODES/mt19937ar.c> was modified by Takahiro Omi as

- delete line 47 "#include<stdio.h>"
- delete line 174 int main(void){...}
- change N -> MT\_N
- change N -> MT\_N
- change the file name "mt19937ar.c" -> "MT.h"

\*/

/\* Period parameters \*/

```
#define MT_N 624
#define MT_M 397
#define MATRIX_A 0x9908b0dfUL /* constant vector a */
#define UPPER_MASK 0x800000000UL /* most significant w-r bits */
#define LOWER_MASK 0x7fffffffUL /* least significant r bits */

static unsigned long mt[MT_N]; /* the array for the state vector */
static int mti=MT_N+1; /* mti==MT_N+1 means mt[MT_N] is not initialized */

/* initializes mt[MT_N] with a seed */
void init_genrand(unsigned long s)
```

```

{
    mt[0]= s & 0xffffffffUL;
    for (mti=1; mti<MT_N; mti++) {
        mt[mti] =
            (1812433253UL * (mt[mti-1] ^ (mt[mti-1] >> 30)) + mti);
        /* See Knuth TAOCP Vol2. 3rd Ed. P.106 for multiplier. */
        /* In the previous versions, MSBs of the seed affect */
        /* only MSBs of the array mt[]. */
        /* 2002/01/09 modified by Makoto Matsumoto */
        mt[mti] &= 0xffffffffUL;
        /* for >32 bit machines */
    }
}

/* initialize by an array with array-length */
/* init_key is the array for initializing keys */
/* key_length is its length */
/* slight change for C++, 2004/2/26 */
void init_by_array(unsigned long init_key[], int key_length)
{
    int i, j, k;
    init_genrand(19650218UL);
    i=1; j=0;
    k = (MT_N>key_length ? MT_N : key_length);
    for (; k; k--) {
        mt[i] = (mt[i] ^ ((mt[i-1] ^ (mt[i-1] >> 30)) * 1664525UL))
            + init_key[j] + j; /* non linear */
        mt[i] &= 0xffffffffUL; /* for WORDSIZE > 32 machines */
        i++; j++;
        if (i>=MT_N) { mt[0] = mt[MT_N-1]; i=1; }
        if (j>=key_length) j=0;
    }
    for (k=MT_N-1; k; k--) {
        mt[i] = (mt[i] ^ ((mt[i-1] ^ (mt[i-1] >> 30)) * 1566083941UL))
            - i; /* non linear */
        mt[i] &= 0xffffffffUL; /* for WORDSIZE > 32 machines */
        i++;
        if (i>=MT_N) { mt[0] = mt[MT_N-1]; i=1; }
    }

    mt[0] = 0x80000000UL; /* MSB is 1; assuring non-zero initial array */
}

/* generates a random number on [0,0xffffffff]-interval */
unsigned long genrand_int32(void)

```

```

{
    unsigned long y;
    static unsigned long mag01[2]={0x0UL, MATRIX_A};
    /* mag01[x] = x * MATRIX_A for x=0,1 */

    if (mti >= MT_N) { /* generate N words at one time */
        int kk;

        if (mti == MT_N+1) /* if init_genrand() has not been called, */
            init_genrand(5489UL); /* a default initial seed is used */

        for (kk=0;kk<MT_N-MT_M;kk++) {
            y = (mt[kk]&UPPER_MASK)|(mt[kk+1]&LOWER_MASK);
            mt[kk] = mt[kk+MT_M] ^ (y >> 1) ^ mag01[y & 0x1UL];
        }
        for (;kk<MT_N-1;kk++) {
            y = (mt[kk]&UPPER_MASK)|(mt[kk+1]&LOWER_MASK);
            mt[kk] = mt[kk+(MT_M-MT_N)] ^ (y >> 1) ^ mag01[y & 0x1UL];
        }
        y = (mt[MT_N-1]&UPPER_MASK)|(mt[0]&LOWER_MASK);
        mt[MT_N-1] = mt[MT_M-1] ^ (y >> 1) ^ mag01[y & 0x1UL];

        mti = 0;
    }

    y = mt[mti++];

    /* Tempering */
    y ^= (y >> 11);
    y ^= (y << 7) & 0x9d2c5680UL;
    y ^= (y << 15) & 0xfc60000UL;
    y ^= (y >> 18);

    return y;
}

/* generates a random number on [0,0xffffffff]-interval */
long genrand_int31(void)
{
    return (long)(genrand_int32()>>1);
}

/* generates a random number on [0,1]-real-interval */
double genrand_real1(void)
{

```

```

    return genrand_int32()*(1.0/4294967295.0);
    /* divided by 2^32-1 */
}

/* generates a random number on [0,1)-real-interval */
double genrand_real2(void)
{
    return genrand_int32()*(1.0/4294967296.0);
    /* divided by 2^32 */
}

/* generates a random number on (0,1)-real-interval */
double genrand_real3(void)
{
    return (((double)genrand_int32()) + 0.5)*(1.0/4294967296.0);
    /* divided by 2^32 */
}

/* generates a random number on [0,1) with 53-bit resolution*/
double genrand_res53(void)
{
    unsigned long a=genrand_int32()>>5, b=genrand_int32()>>6;
    return(a*67108864.0+b)*(1.0/9007199254740992.0);
}
/* These real versions are due to Isaku Wada, 2002/01/09 added */

```

### calculation\_game3.c

```

*****
*
*
* バージョンアップ版です。
* 亂数に偏りがあるので、
* メルセンヌツイスター (Mersenne Twister) 法を用いています。
* http://www.satt.u-tokyo.ac.jp/~omi/random_variables_generation.html
* 綺麗な乱数が得られることに着目して下さい。
*
* 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。
*
*****
*/
#include <stdio.h>
#include <math.h>

```

```

#include <string.h>
#include <time.h>

#include "mt.h" // Mersenne Twister法による乱数
// 以下の関数が用意されている。
// genrand_int32() // 符号なし32ビット長整数
// genrand_int31() // 符号なし31ビット長整数
// genrand_real1() // 一様実乱数[0,1] (32ビット精度)
// genrand_real2() // 一様実乱数[0,1] (32ビット精度)
// genrand_real3() // 一様実乱数(0,1) (32ビット精度)
// genrand_res53() // 一様実乱数[0,1] (53ビット精度)
// AからBの範囲の整数の乱数が欲しいときには
// genrand_int32()% (B-A+1)+A;
// のような関数を用いればよい。

int main(int argc, char const *argv[]) {

    int      operand1;           // 第一被演算子
    int      operand2;           // 第二被演算子
    char     operator;          // 演算子 ('+', '-', '*', '/')
    int      digit;             // 桁数
    int      operation_result;  // 演算結果
    int      your_answer;       // 回答
    int      correct_answer;    // 正答数
    time_t   start_time;        // ゲーム開始時刻
    time_t   finish_time;       // ゲーム完了時刻
    int      i;

    // オープニング
    printf("*****\n");
    printf("*\n");
    printf("*      計算ゲームへようこそ\n");
    printf("*\n");
    printf("*****\n");
    printf("\n");

    // 桁数をリクエスト
    do {
        printf("何桁の数字で挑戦しますか？ \n");
        printf("一桁: 1    二桁: 2    三桁: 3    四桁: 4 \n");
        scanf("%d", &digit);
        while(getchar() != '\n'); // キーバッファ読み飛ばす
    } while (digit <= 0 || digit >= 5);

    // 演算子をリクエスト
}

```

```

do {
    printf("どの計算に挑戦しますか？ \n");
    printf("加算: '+' 減算: '-' 乗算: '*' 除算: '/' \n");
    scanf("%c", &operator);
    while(getchar() != '\n'); // キーバッファ読み飛ばす
} while (operator != '+' && operator != '-' && operator != '*' && operator != '/');

// 出題処理
correct_answer = 0; // 正答数初期化
time(&start_time); // ゲーム開始時刻の保存
for (i = 1; i <= 10; i++){
    // digit桁の乱数を求める
    // genrand_int32 は 32bit(0-約43億) の整数型の乱数を返すので、
    // 10(または100, 1000, 10000)で割った余りが得たい乱数となる。
    // (int) は、キャストと呼ばれる。
    // pow関数の戻り値はdouble型なので、int型へ変換している。
    if (operator != '/') {
        operand1 = genrand_int32() % (int)pow(10, digit);
    } else {
        // 除算時、同じ桁同士で演算すると、ほぼ0となるので、調整。
        operand1 = genrand_int32() % (int)pow(10, digit+1);
    }
    do {
        operand2 = genrand_int32() % (int)pow(10, digit);
    } while (operator == '/' && operand2 == 0); // 0 の除算は定義されて
    // いない

    // 正答を用意
    switch(operator){
        case '+': operation_result = operand1 + operand2; break;
        case '-': operation_result = operand1 - operand2; break;
        case '*': operation_result = operand1 * operand2; break;
        case '/': operation_result = operand1 / operand2; break;
    }

    // 何回目のゲームか、表示
    switch(operator){
        case '+': printf("足し算ゲーム %d 回目\n", i); break;
        case '-': printf("引き算ゲーム %d 回目\n", i); break;
        case '*': printf("掛け算ゲーム %d 回目\n", i); break;
        case '/': printf("割り算ゲーム %d 回目\n", i); break;
    }

    // 出題する
}

```

```

printf("%d %c %d = ?\n", operand1, operator, operand2);

// 回答を受け取る
scanf("%d", &your_answer);
while(getchar() != '\n'); // キーバッファ読み飛ばす

// 正解発表と正答数のカウント
if (your_answer == operation_result){
    printf("正解です。\n");
    correct_answer++;
} else {
    printf("正解は、%d です。\n", operation_result);
}
// ゲーム完了時刻の保存
time(&finish_time);

// 総合結果発表
printf("*****\n");
printf("*\n");
printf("*      結 果 発 表\n");
printf("*\n");
printf("*      10問中 %d 問 正解\n", correct_answer);
printf("*      %ld 秒 でクリア！\n", finish_time - start_time);
printf("*\n");
printf("*      おめでとうございます！\n");
printf("*\n");
printf("*****\n");
printf("\n");

return 0;
}

```

## 8.BMI.

[bmi.c](#)

```

*****
*
*
* BMI を算出
*
*****
*/

```

```
#include <stdio.h>
#include <math.h>

int main(int argc, char const *argv[]) {

    // 変数宣言
    int height; // 身長
    int weight; // 体重
    double bmi; // BMI

    // 入力処理
    printf("身長(cm)を入力して下さい。\\n");
    scanf("%d", &height);
    while(getchar() != '\\n');

    printf("体重(kg)を入力して下さい。\\n");
    scanf("%d", &weight);
    while(getchar() != '\\n');

    // BMI算出
    // int型のheightを、キャスト演算子(double)を使って、
    // double型に変換しています。
    // 優先順位を明確にするため、
    // ((double)height) を丸括弧で囲ってから、
    // 100 で割っている点に着目して下さい。
    bmi = weight / pow( ((double)height) / 100, 2);

    // 結果表示
    printf("BMI は %4.2f です。\\n", bmi);
    if (bmi < 18.5){
        printf("痩せすぎです。\\n");
    } else if (bmi < 25) {
        printf("標準です。\\n");
    } else {
        printf("肥満です。\\n");
    }

    return 0;
}
```

## 9.閏年.

[leap\\_year.c](#)

```
*****
```

```
**
*
* 閏年を算出します。
*
*****
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

// 分かりやすさのために、TRUE, FALSE という定数を定義します。
// 定数は全て大文字で書く慣習です。
#define TRUE 1
#define FALSE 0

// 関数のプロトタイプ宣言
// 西暦年を渡して、閏年なら、TRUEを返す関数
int leap_year1(int year);
int leap_year2(int year);

int main(int argc, char const *argv[]) {
    // コマンドラインから、引数を渡すことも出来ます。
    if (argc == 1) {
        printf("【使い方】\n");
        printf("閏年(leap year)か、平年(common year)か、算出するプログラムです。\n");
        printf("%s 2016\n", argv[0]); // argv[0] はプログラム自身の名前です。
        printf("の様に西暦年で入力して下さい。.\n");

        exit(1); // プログラムを終了します。
    }

    int year; // 西暦年

    char *endptr; // 数値に変換出来なかった文字
    // 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。
    // 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、
    // 区別が出来ないので、strtol関数を使います。
    year = strtol(argv[1], &endptr, 10); // 10進数として変換する

    // 数値変換不可の文字列の長さを調べる。
    if (strlen(endptr) != 0){
        // エラーメッセージ出力
        printf("%s は、西暦年として認識出来ませんでした。.\n", argv[1]);
    }
}
```

```

    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。
}

// 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。
// leap_year1 : 長いif文の関数
if (leap_year1(year) == TRUE) {
    printf("閏年です。\\n");
} else {
    printf("平年です。\\n");
}

// leap_year2 : 整理したif文の関数
if (leap_year2(year) == TRUE) {
    printf("閏年です。\\n");
} else {
    printf("平年です。\\n");
}

return 0;
} // main()関数の最後です。

```

```

// 西暦年を渡して、閏年なら、TRUEを返す関数
int leap_year1(int year){
    // 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。

    // 素直にif文で書くと次のようになります。
    // (if文の中にif文を書くことも出来ます。)
    if (year % 4 == 0){
        // 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。
        if (year % 100 == 0){
            // 但し100で割り切れる年は閏年ではありません。とありますが、
            // さらにこれの例外があるので、例外を書きます。
            if (year % 400 == 0){
                // しかしながら、400で割り切れる年は、閏年です。とあるので、
                // TRUEを返します。
                return TRUE;
            } else {
                // 400で割り切れなかった年です。
                return FALSE;
            }
        } else {
            // 100で割り切れなかった年です。
        }
    }
}

```

```

        return TRUE;
    }
} else {
    // 4で割り切れなかった年です。
    return FALSE;
}
}

// 西暦年を渡して、閏年なら、TRUEを返す関数
int leap_year2(int year){
    // 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。

    // 素直にif文を書くとすごく長くなってしまいました。
    // 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、
    // 確認するのも大変です。

    // 4で割り切れる.....(a)
    // 100で割り切れる.....(b)
    // 400で割り切れる.....(c)
    // と条件が複合しているので、
    // 閏年かどうか.....(x)
    // 表にして整理すると、分かりやすいです。
    // (真偽値表、カルノー図と言います)

    // 割り切ることを TRUE(T)
    // 割り切れないことを FALSE(F) として、表にしてみましょう。
    // 8とおりの組み合わせが出来ます。
    // - 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。
    // 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。

    // a b c x
    // F - - F ... (1)
    // F - - F ... (2)
    // F - - F ... (3)
    // F - - F ... (4)
    // T F - T ... (5)
    // T F - T ... (6)
    // T T F F ... (7)
    // T T T T ... (8)

    // 表を見ると、(5) と (6) は同じですので、
    // (5) または (8)の場合で、閏年になることが分かります。
}

```

```
// つまり、
// 4で割り切れて、100で割り切れない年
// または、
// 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)
// の場合に閏年になることが分かります。

// よって次のif文でよいことが分かります。
if ( (year % 4 == 0 && year % 100 != 0) || year % 400 == 0 ){
    return TRUE;
} else {
    return FALSE;
}
// 1600年, 1700年, 2000年, 2004年, 2017年で、
// を入れて、確認してみましょう。
// (どのテストケースで確認すべきか、考えるのも大切です)
}
```

## 10.算用数字を漢数字に変換.

## num2kan.c

```
/*
**
*
* 算用数字 302 を 漢数字にします。
* 無量大数までの変換が出来ます。
* 参考: https://ja.wikipedia.org/wiki/命数法
*
* 正規表現については、
* https://ja.wikipedia.org/wiki/正規表現
* http://rubular.com 参照
*
*****
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <regex.h> // 正規表現 regcomp, regex

// 漢字変換に用いる定数
// 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
// const は 変更禁止の為の修飾子
const char *digit[] = { "〇", "一", "二", "三", "四", "五", "六", "七", "八",
    "九" };
const char *unit1[] = { "千", "百", "十", "" };
```

```

const char *unit2[] = { "", "万", "億", "兆", "京", "垓", "秭",
                      "穰", "溝", "澗", "正", "載", "極",
                      "恒河沙", "阿僧祇", "那由他", "不可思議", "無量大数"
};

// 4桁の漢数字への変換を行う関数
char *num2kan(char *str);

int main(int argc, char const *argv[]) {

    // コマンドラインからの引数がなければ、使い方表示
    if (argc == 1) {
        printf("【使い方】\n");
        printf("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");
        printf("%s 302\n", argv[0]);
        printf("の様に入力して下さい。 \n");
        exit(1);
    }

    // コマンドライン引数チェック
    // '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。
    // (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 )

    // 正規表現のC言語でのコーディングについては、
    // http://atomic.jpn.ph/prog/lang/regex.html 参照

    char *reg = "[0-9]+";    // 全て数字であるか判定するための正規表現
    regex_t regist;
    // 正規表現のコンパイル
    if (regcomp(&regist, reg, REG_EXTENDED)) { return 1; }
    // argv[1]が、正規表現にマッチするか、実行
    if (regexec(&regist, argv[1], 0, NULL, 0)) { // マッチしない場合
        printf("%s は 算用数字として認識出来ませんでした。 \n", argv[1]);
        exit(1);
    }
    // コンパイル結果の開放は必須
    regfree(&regist);

    // 最大取り扱い桁数 9999無量大数( $10^{68}$ )までの数
    if (strlen(argv[1]) > 4+68+1){
        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]);
        exit(1);
    }

    // 漢数字への変換処理
}

```

```

// 4桁ごとに漢数字に変換する
// 例)    12      3456          7890
// => 十二 億 三千四百五十六 万 七千八百九拾

// 変数宣言
int digit_length;           // 何桁の数字か
int i, u;
char string_number[4+68+1+1]; // 算用数字 最大取り扱い桁数 9999無量
                               // 大数(9999*1e68)まで
strcpy(string_number, "");   // 初期化
// 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
char chinese_numeral[(7*3)*(72/4) + sizeof(unit2)+1];
strcpy(chinese_numeral, "");
char work[7*3+1];
strcpy(work, "");

// 4桁ごとに区切って渡せるよう、先頭に0を付与する
// 12 3456 7890 => 0012 3456 7890
digit_length = strlen(argv[1]); // 何桁の数であるか？
int giving_zero = 4 - (digit_length % 4); // 先頭に付与すべき0の数
if (giving_zero == 4) { giving_zero = 0; };
for (i = 0; i < giving_zero; i++) {
    strcat(string_number, "0");
}
strcat(string_number, argv[1]);

// 何桁の数字か
digit_length = strlen(string_number);

// 4桁毎に漢数字に変換
// 4桁までなら unit2 "" // 8桁までなら unit2 "万" // 12桁までなら unit2 "億"
// を付与する
int unit2_index = (digit_length / 4) - 1;
for (u = 0; u < digit_length/4; u++, unit2_index--){
    // string_number から 各ユニット毎に、4文字ずつ work にコピー
    work[0] = string_number[4*u + 0];
    work[1] = string_number[4*u + 1];
    work[2] = string_number[4*u + 2];
    work[3] = string_number[4*u + 3];
    work[4] = '\n';

    // 4桁毎に漢数字に変換、結果を連結する
    strcat(chinese_numeral, num2kan(work));
    // ... "兆", "億", "万", "" の付与
    strcat(chinese_numeral, unit2[unit2_index]);
}

```

```
}

// 0 の場合のみ、空文字列のままなので、"〇" にする。
if (strlen(chinese_numeral) == 0){
    strcpy(chinese_numeral, digit[0]);
}

// 結果表示
printf("%s\n", chinese_numeral);

return 0;
}

// 4桁 の 漢数字へ変換する関数
char *num2kan(char *str){
    // 変数宣言
    int i;
    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
    strcpy(work, ""); // 初期化

    // 一つずつ漢数字に変換する
    for(i = 0; i < 4; i++){
        switch(str[i]){
            case '0':
                break; // 何もせず、次の桁へ進む
            case '1':
                // 一の位のみ "一"と書く
                if (i == 3) {
                    strcat(work, digit[str[i]-'0']); // str[i]-'0' 文字0から数値0へ変換
                }
                strcat(work, unit1[i]); // "千", "百", "十", "" のいずれか
                break;
            default:
                strcat(work, digit[str[i]-'0']); // "一" ~ "九"
                strcat(work, unit1[i]); // "千", "百", "十", "" のいずれか
                break;
        }
    }

    // 作業結果を戻す
    return strcpy(str, work);
}
```

## 解答例 ★★★★☆☆☆☆

1.1 10人の名前を点数順に並び替えて表示。

[score\\_and\\_name\\_sort.c](#)

```
*****
*
*
* 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、クイックソート(quicksort)と呼ばれるアルゴリズムで
* 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。
* (前回、作成したプログラムを流用してももちろんOKです)
*
* クイックソートに関しては、
* https://ja.wikipedia.org/wiki/クイックソート
* を参照して下さい。
*
* qsortの利用方法については、
* http://www.cc.kyoto-su.ac.jp/~yamada/ap/qsort.html より、引用
*
*****
```

---

```
/
```

```
#include <stdio.h>
#include <stdlib.h> // quicksort
#include <string.h>
```

```
// 比較関数（大小判断を返す関数）
int compare_int(const void *a, const void *b){
    // b > a なら 正の数を返す。(降順)
    return *(int*)b - *(int*)a;
}
```

```
int main(int argc, char const *argv[]) {
```

```
    // 変数宣言
    char *name[] = { "亜希子", "加世子", "小夜子", "妙子", "奈美子", "太郎", "次郎",
    "三郎", "四郎", "五郎" };
    int score[] = { 99,           88,           77,           66,           55,
    55,           64,           73,           82,           91 };
```

```
int backup[10];
int i, j, s;

// 結果表示
printf("並び替え前:\n");
for (i = 0; i < 10; i++){
    printf("%d %s \n", score[i], name[i]);
}

// 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
// for 文で 一要素ずつコピーしても良いが、
// memcpy 関数を紹介
memcpy(backup, score, sizeof(int)*10); // int型10要素分をbackupへコピー

// 確認用
// printf("score backup\n");
// for (i = 0; i < 10; i++){
//     printf("%d %d \n", score[i], backup[i]);
// }

// 並び替えを行う
// 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
qsort(score, 10, sizeof(int), compare_int);

// 確認用
// printf("並び替え後:\n");
// for (i = 0; i < 10; i++){
//     printf("%d\n", score[i]);
// }

printf("並び替え後:\n");
// 並び替え結果に基づいて、名前を表示する
for (i = 0; i < 10; i){
    // 例) i = 1 のとき s には 91 点が入っている
    // 名前も連動して並び替えられると良いが、
    // そういう作りにはしていないので、
    // backup配列を参照して、
    // 91 点の人は何番目であったか、
    // 検索する
    s = score[i];
    for (j = 0; j < 10; j++) {
        if (s == backup[j]) {
            // 同じ点数（奈美子と太郎）の場合、太郎が表示されなくなることを防ぐため、
            // あり得ない点数の -1 を設定する。
            // （「番兵」と呼ばれる）
```

```

        backup[j] = -1;
        break; // 元の順番では、j 番目であったことが分かった
    }
}
printf("%d %s\n", score[i], name[j]);
}

return 0;
}

```

## 2.素数.

[prime\\_number.c](#)

```

*****
*
*
* 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？
*
*****
/
#include <stdio.h>

#define TRUE 1
#define FALSE 0

// 素数判定関数
int is_prime(int candidate, int prime_array[], int prime_index);

int main(int argc, char const *argv[]) {

    // 素数の数 pi(x) ~= x / log(x) が知られている。
    // なので1000までの素数の数は、たかだか2000 である。
    // よって、要素数2000とする
    int prime_array[2000] = {};

    // = {} 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
    // for (int i = 0; i < 2000; i++){
    //     prime_array[i] = 0;
    // }

    int prime_index = 0;
    int i;

```

```

// 2 は 偶数唯一の素数である。
prime_array[0] = 2;
prime_index = 1;

int candidate = 3; // 素数候補 3から始める

while (candidate < 10000){
    // 素数かどうかの判定は、is_prime関数に任せ、
    // 素数であったら、配列に追加する
    if (is_prime(candidate, prime_array, prime_index)){
        prime_array[prime_index] = candidate;
        prime_index++;
        candidate += 2; // 奇数の候補のみ調べるので、+2 している
    } else {
        candidate += 2;
    }
}

// 結果表示
// (せっかくなので、10000までの素数表示)
for (int i = 0; i < prime_index; i++){
    if (i % 10 == 0){
        printf("%5d: ", i + 1);
    }
    if (prime_array[i] != 0){
        printf("%5d ", prime_array[i]);
    }
    // 10 個 ずつ 改行
    if ((i + 1) % 10 == 0){
        printf("\n");
    }
}
printf("\n");

printf("1000番目の素数は、%d です。\\n", prime_array[999]);
}

// 素数判定関数
int is_prime(int candidate, int prime_array[], int prime_index){
    int i;
    int precious_metal; // 素数でないにしても、貴金属ではある。

    // 10000(=100*100) が 素数かどうかを調べるには、
    // 100までの素数で割りきれるかどうか、確認すれば良い。

```

```
// 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。
for (i = 0; i < prime_index; i++){
    precious_metal = prime_array[i];           // 割り切れるかどうか
    if (candidate % precious_metal == 0){
        // 割り切れたなら、素数ではない。
        return FALSE;
    }
}
return TRUE;
```

## 3.完全数.

[perfect\\_number.c](#)

```
/*
*
*
* 完全数とは次のような数のことです。
* 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。
* 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？
*
*/
#include <stdio.h>

#define TRUE 1
#define FALSE 0

// 完全数判定関数
int is_perfect(int candidate);

int main(int argc, char const *argv[]) {
    // 6 = 2 * 3
    // 28 =  $2^2 \times 7$  である
    // 割り切れるかどうかを調べ、
    // 割り切った合計が、もとの数と一致するか調べればよい。

    int perfect_array[4] = { 6, 28 };
    int perfect_index = 2;

    int candidate = 28 + 2; // 奇数の完全数は知られていないため、
```

// 28 の次の偶数を候補と

する

```

while(1){
    // 完全数かどうかの判定は、is_perfect関数に任せ、
    // 完全数が発見されるまで、繰り返す
    if (is_perfect(candidate)){
        // 完全数が発見されたら追加
        perfect_array[perfect_index] = candidate;
        perfect_index++;
        if (perfect_index == 4){
            break;
        } else {
            candidate += 2;
        }
    } else {
        candidate += 2;
    }
}

// 結果表示
for (int i = 0; i < perfect_index; i++){
    printf(" %d 番目の完全数は、%5d です。\\n", i+1, perfect_array[i]);
}
}

// 完全数判定関数
int is_perfect(int candidate){
    int sum = 0;      // 総和
    int i;

    for (i = 1; i < candidate; i++){
        if (candidate % i == 0){
            sum += i;
        }
    }
    if (sum == candidate) {
        return TRUE;
    } else {
        return FALSE;
    }
}

```

[word\\_book.c](#)

```
*****
*
*
* 英単語帳アプリを創ってみましょう。
* I -> わたし
* love -> 愛する
* you -> あなた
* と答えられたら、満点です。
* 10問出題して、英語力向上を目指しましょう。
*
*****
/
#include <stdio.h>
#include "mt.h"

#define CHOICES_SIZE 3 // 三択で出題する
#define TRUE 1
#define FALSE 0

int main(int argc, char const *argv[]) {

    // 出題用配列
    char *english_words[] = { "", "I", "love", "you", "C", "language", "lesson", "happy", "hacker", "programming", "computer" };
    char *japanese_words[] = { "", "わたし", "愛する", "あなた", "C", "言語", "学習", "幸せ", "技術者", "プログラミング", "コンピュータ" };

    // 三択で出題する
    int choices[CHOICES_SIZE+1]; // 0 は未使用とするため、+1
    int candidate; // 選択肢の候補
    int registered_word = 10; // 登録単語数
    int question_number; // 第何問目か？
    int correct_answer; // 正答
    int your_answer; // 入力された回答
    int score; // 得点
    int flag;
    int i;

    // オープニング
    printf("*****\n");
    printf("*\n");

```

```

printf("*          英単語ゲームへようこそ          \n");
printf("*                                \n");
printf("*****\n");
printf("\n");

// 出題処理

// 10題出題する
score = 0;
for (question_number = 1; question_number <= 10; question_number++) {
    printf("【第 %d 問】\n", question_number);

    // 選択肢の初期化
    for (i = 0; i <= CHOICES_SIZE; i++) {
        choices[i] = 0;
    }

    // choices[1], [2], [3] に出題番号をセット
    do {
        // 1から3の乱数を取得
        candidate = genrand_int32() % registerd_word + 1;
        // 選択肢に登録済みか調べる。
        int flag = FALSE;
        for (i = 1; i <= CHOICES_SIZE; i++) {
            if (candidate == choices[i]){
                flag = TRUE; // すでに選ばれていた
                break;
            }
        }
        // 選択肢には未登録だったので、登録する
        if (flag == FALSE){
            for (i = 1; i <= CHOICES_SIZE ; i++) {
                if (choices[i] == 0){
                    // i番目の選択肢として登録する
                    choices[i] = candidate;
                    break;
                }
            }
        }
    } while(choices[CHOICES_SIZE] == 0);

    // 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
} while(choices[CHOICES_SIZE] == 0);

// choice[1], [2], [3] の中から、いずれかを正解として設定する
correct_answer = genrand_int32() % 3 + 1;

```

```

// 出題する
printf("%s\n", english_words[choices[correct_answer]]);

// 選択肢を提示する
for (i = 1; i <= CHOICES_SIZE; i++){
    printf("%d: %s ", i, japanese_words[choices[i]]);
}
printf("\n");

// 1-3までの入力を求める
do {
    scanf("%d", &your_answer);
    while(getchar() != '\n'); // キーバッファ読み飛ばす
} while (your_answer <= 0 || your_answer > CHOICES_SIZE);

// 正解判定
if (your_answer == correct_answer) {
    printf("正解です！\n\n");
    score++; // 得点加算
} else {
    printf("残念。正解は、%d: %s です。\n\n", correct_answer, japanese_words
[choices[correct_answer]]);
}

// 総合結果発表
printf("*****\n");
printf("*\n");
printf("          結 果 発 表\n");
printf("*\n");
printf("          10 問中 %d 問 正解\n", score);
printf("*\n");
printf("          おめでとうございます！\n");
printf("*\n");
printf("*****\n");
printf("\n");

return 0;
}

```

## 5. すごろく.

sugoroku.c

```
*****
```

```
*  
*  
* 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。 *  
*  
*****  
/  
  
#include <stdio.h>  
#include <string.h>  
#include "mt.h"  
  
#define MAP_SIZE      22 // 0-21までの升目がある  
#define GOAL_POSITION 21 // 21の升目が、上がり  
#define TRUE   1  
#define FALSE  0  
  
/* 双六のマップ配置  
0:スタート  
1:  
2:  
3:  
4:2マス進む  
5:3マス戻る  
6:  
7:スタートに戻る  
8:2マス進む  
9:1回休み  
10:3マス戻る  
11:  
12:2マス進む  
13:  
14:スタートに戻る  
15:3マス戻る  
16:2マス進む  
17:  
18:2回休み  
19:  
20:3マス戻る  
21:ゴール  
 */  
  
// main の外側に配置 (グローバル変数)
```

```

// 各関数から共通して見られるようとする
// (一般的には好ましくないが、双六作成が容易となるため)
char const *map_event[] = { "", "", "", "", "2A", "3B", "",
                           "SB", "", "1R", "3B", "", "2A", "",
                           "SB", "3B", "2A", "", "2R", "", "3B", "" };

// 列挙型 cmp(競技者)型の宣言
typedef enum { PLAYER, COMPUTER } cmp;
// cmp型変数 competitorの宣言
cmp competitor;
// #define PLAYER 0
// #define COMPUTER 1 と同じ
// enum のご紹介のみ

// PLAYER, COMPUTER それぞれが、双六上のどの升目にいるか？
int position[2] = {};
// PLAYER, COMPUTER 何回休みか？
char rest[2];

// 関数のプロトタイプ宣言 省略するため
// main 関数を末尾に記載

// 双六表示機能
void draw_map(){
    int i;
    printf("\n");
    printf(" ST  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 2
0 GL \n");
    printf("+-+-+-+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+\n");
    //player
    for (i = 0; i < MAP_SIZE; i++){
        printf("| ");
        if (position[PLAYER] == i){
            printf("P");
        } else {
            printf(" ");
        }
    }
    printf("|\n");
    //computer
    for (i = 0; i < MAP_SIZE; i++){
        printf("| ");
        if (position[COMPUTER] == i){

```



```

;
    puts("# #####     #####     ###     ##### #     # ##### #     # ##### #")
;
    puts("#                                     #")
;
    puts("#####")
;
    puts("");
}

void drow_ending() {
    puts("");
    puts("#####")
;
    puts("#                                     #")
;
    puts("#      The Japanese Traditional Game          #")
;
    puts("#                                     S U G O R O K U          #");
    puts("#                                     #")
;
    puts("#                                     Fin.          #")
;
    puts("#                                     #")
;
    puts("#####")
;
    puts("");
}
;

// マップ上のイベントを受け取り、競技者の内部状態を適宜変更する
// " |   |   |   |2A|3B|   |SB|   |1R|3B|   |2A|   |SB|3B|2A|   |2R|   |3B|   |\n"
);
// " legend: nV. n: times V: verb. go Ahead / go Backward / Rest           \n"
);
void event(cmp competitor){
    if (position[competitor] > GOAL_POSITION){
        position[competitor] = GOAL_POSITION;
    }

    // イベントを取得
    char event[3];
    strcpy(event, map_event[position[competitor]]));
}

```

```

if (strlen(event) == 0){
    // 何もイベントはなかったとして終了
    return;
}

// 該当イベントの処理
char times = event[0]; // 何升進む、何回休みなどの数
char kind = event[1]; // 進む、戻る、休むの種類
switch (kind){
case 'A': // 進む
    if (times == 'S'){
        // 振り出しに戻る
        position[competitor] = 0;
    } else {
        // 進める
        position[competitor] += (times - '0'); // 文字を数値に変換
        // ゴールを通り過ぎていたら、ゴール地点に調整
        if (position[competitor] > GOAL_POSITION){
            position[competitor] = GOAL_POSITION;
        }
    }
    // メッセージ表示
    if (competitor == PLAYER){
        printf("P> やった～ %d マス 進んだ！\n", times - '0');
    } else {
        printf("C> やった～ %d マス 進んだ！\n", times - '0');
    }
    break;
case 'B': // 戻る
    if (times == 'S'){
        // 振り出しに戻る
        position[competitor] = 0;
    } else {
        // 戻る
        position[competitor] -= (times - '0'); // 文字を数値に変換
        // スタートを通り過ぎていたら、スタート地点に調整
        if (position[competitor] < 0) {
            position[competitor] = 0;
        }
    }
    // メッセージ表示
    if (times == 'S'){
        if (competitor == PLAYER){
            printf("P> わ～ん スタートに 戻ったよ\n");
        } else {
    
```

```

        printf("C> わ～ん スタートに 戻ったよ\n");
    }
} else {
    if (competitor == PLAYER){
        printf("P> わ～ん %d マス 戻ったよ\n", times - '0');
    } else {
        printf("C> わ～ん %d マス 戻ったよ\n", times - '0');
    }
}
break;
case 'R': // お休み
rest[competitor] = (times - '0'); // 文字を数値に変換
// メッセージ表示
if (competitor == PLAYER){
    printf("P> わ～ん %d 回 休みだよ\n", times - '0');
} else {
    printf("C> わ～ん %d 回 休みだよ\n", times - '0');
}
break;
}

// さいころを振って進む
void dice_and_walk(cmp competitor){
    int dice = genrand_int32() % 6 + 1;
    position[competitor] += dice;
// ゴールを通り過ぎていたら、ゴール地点に調整
    if (position[competitor] > GOAL_POSITION){
        position[competitor] = GOAL_POSITION;
    }
// メッセージ表示
    if (competitor == PLAYER){
        printf("P> やった～ %d マス 進んだ！\n", dice);
    } else {
        printf("C> やった～ %d マス 進んだ！\n", dice);
    }
}

// 双六を上がったか、判定関数
int is_goal(cmp competitor){
    if (position[competitor] == GOAL_POSITION){
        return TRUE;
    } else {
        return FALSE;
    }
}

```

```
}
```

```
int main(int argc, char const *argv[]) {
```

```
    // タイトル表示
```

```
    draw_opening();
```

```
    // 双六表示
```

```
    draw_map();
```

```
    printf("\nPress Enter key\n");
```

```
    while(getchar() != '\n');
```

```
    // 競技開始
```

```
    // どちらかが上がるまで、続行
```

```
    do {
```

```
        // PLAYERの番
```

```
        printf("\nPLAYERの番 Press Enter key\n");
```

```
        while(getchar() != '\n');
```

```
        if (rest[PLAYER] == 0){
```

```
            // ○回休みでなければ、さいころを振って進む
```

```
            dice_and_walk(PLAYER);
```

```
            // 止まった升目にないかイベントが設定されているか
```

```
            event(PLAYER);
```

```
        } else {
```

```
            // お休み回数を減らす
```

```
            printf("P> %d 回休みなので進めない・・・\n", rest[PLAYER]);
```

```
            rest[PLAYER]--;
```

```
        }
```

```
        // 双六表示
```

```
        draw_map();
```

```
        // COMPUTERの番
```

```
        printf("\nCOMPUTERの番 Press Enter key\n");
```

```
        while(getchar() != '\n');
```

```
        if (rest[COMPUTER] == 0){
```

```
            // ○回休みでなければ、さいころを振って進む
```

```
            dice_and_walk(COMPUTER);
```

```
            // 止まった升目にないかイベントが設定されているか
```

```
            event(COMPUTER);
```

```
        } else {
```

```
// お休み回数を減らす
printf("C> %d 回休みなので進めない・・・\n", rest[COMPUTER]);
rest[COMPUTER]--;
}

// 双六表示
draw_map();

} while(!is_goal(PLAYER) && !is_goal(COMPUTER));

// ゴール到着時のメッセージ
if (is_goal(PLAYER)){
    printf("P> 双六を上がったよ(*^_^*)\n");
} else {
    printf("C> 双六を上がったよ(*^_^*)\n");
}

// エンディング表示
draw_ending();

return 0;
}
```

## 解答例 ★★★★★☆☆

1.今日までに何日生きた.

[lifetime.c](#)

```
*****
*
*
* 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？
*
*****
/
#include <stdio.h>
#include <time.h> // 日付時刻の操作用関数
#include <math.h>
#include <string.h>
#include <stdlib.h>

// 年月日からユリウス通日を求める関数
int julian_day_number(int year, int month, int day);
// ユリウス通日から年月日を返す関数
void inverse_julian_day_number(int jdn, int *year, int *month, int *day);

int main(int argc, char const *argv[]) {

    // コマンドラインからの引数がなければ、使い方表示
    if (argc == 1) {
        printf("【使い方】\n");
        printf("今日まで何日来たかを表示します。\n");
        printf("1980年1月1日生まれであれば、\n");
        printf("%s 19800101\n", argv[0]);
        printf("昭和20年8月10日生まれであれば、\n");
        printf("%s 3200810\n", argv[0]);
        printf("の様に入力して下さい。 \n");
        exit(1);
    }
}
```

```

int birthday; // 生年月日
char *endptr; // 数値に変換出来なかった文字
int b_year; // 誕生日の西暦年
int b_month; // 誕生日の月
int b_day; // 誕生日の日

// 簡易エラーチェック
// 数値変換不可の文字列の長さを調べる。
birthday = strtol(argv[1], &endptr, 10); // 10進数として変換する
if (strlen(endptr) != 0){
    // エラーメッセージ出力
    printf("%s は、生年月日として認識出来ませんでした。\\n", endptr);
    exit(1);
}

// 生年月日に分離する
// 年
// 元号入力なら、西暦へ変換
if (birthday < 1e7){
    int ge = birthday / 10000; // 上3桁 元号符号+元号年
    switch (ge/100) {
        case 1: b_year = ge % 100 + 1867; break;
        case 2: b_year = ge % 100 + 1911; break;
        case 3: b_year = ge % 100 + 1925; break;
        case 4: b_year = ge % 100 + 1988; break;
    }
} else {
    b_year = birthday / 10000; // 上4桁は西暦年
}
// 月
b_month = birthday / 100; // 下二桁を切り捨てて
b_month = b_month % 100; // 残った数の下2桁

// 日
b_day = birthday % 100; // 下2桁は日

// http://simd.jugem.jp/?eid=149 より引用
// 今日の日付を取得する
time_t now;
struct tm *ltm;
time( &now );
ltm = localtime( &now );

```

```

// 確認用

// printf( "%5d : [年]\n", ltm->tm_year + 1900 );
// printf( "%5d : [月]\n", ltm->tm_mon + 1 );
// printf( "%5d : [日]\n", ltm->tm_mday );
// printf( "%5d : [時]\n", ltm->tm_hour );
// printf( "%5d : [分]\n", ltm->tm_min );
// printf( "%5d : [秒]\n", ltm->tm_sec );
// printf( "%5d : [曜日]\n", ltm->tm_wday );
// printf( "%5d : [経過日数]\n", ltm->tm_yday );
// printf( "%5d : [夏時間の有無]\n", ltm->tm_isdst );

int t_year = ltm->tm_year + 1900;           // 今日の西暦年
int t_month = ltm->tm_mon + 1;               // 今日の月
int t_day = ltm->tm_mday;                   // 今日の日

// strftime関数を用いて文字列にすることも可能
// char str_time[100];
// int maxsize = 100;
// char *format = "%Y年%m月%d日 %H:%M";
// strftime(str_time, maxsize, format, ltm);
// printf("%s\n", str_time);

// まともに暦の日付計算をしようとするとき大変なので、
// ある基準日からの経過日数を求めることにします。
// 今日は、 基準日から何日目
// 誕生日は、基準日から何日目 と分かれば、
// 引き算することで、日数を計算出来ます。
// ユリウス通日（つうじつ）として知られており、
// 紀元前4713年1月1日 を第一日として、
// 天文学などの分野で用いられています。
// https://ja.wikipedia.org/wiki/ユリウス通日
// に公式がありますので、プログラミングしましたが、
// http://ufcpp.net/study/algorithm/o\_days.html
// にも説明があります。

printf("今日は      %4d 年 %2d 月 %2d 日です。 \n", t_year, t_month, t_day);
printf("誕生日は      %4d 年 %2d 月 %2d 日です。 \n", b_year, b_month, b_day);

// 今日のユリウス通日
int today_jdn = julian_day_number(t_year, t_month, t_day);
// printf("今日の ユリウス通日 %d\n", today_jdn);
int birthday_jdn = julian_day_number(b_year, b_month, b_day);
// printf("誕生日の ユリウス通日 %d\n", birthday_jdn);

// 結果表示

```

```

printf("今日は生後 %5d 日 です。\\n", today_jdn - birthday_jdn);

// 生まれてから30000日後がいつかを求めます。
int life_30000_jdn = birthday_jdn + 30000;
int l_year, l_month, l_day;
// 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。
inverse_julian_day_number(life_30000_jdn, &l_year, &l_month, &l_day);
printf("生後三万日は %4d 年 %2d 月 %2d 日 です。\\n", l_year, l_month, l_day)
;

return 0;
}

// 年月日からユリウス通日を求める関数
int julian_day_number(int year, int month, int day){
/*
https://ja.wikipedia.org/wiki/ユリウス通日
// 2月のユリウス通日が不正

int y, m, d;
int n;
int mjd;
int jdn;

y = year + (month - 3) / 12;
m = (month - 3) % 12;
d = day - 1;
n = d + (153*m+2)/5 + 365*y + y/4 - y/100 + y/400;
mjd = n - 678881;
jdn = mjd + 2400001;
*/
int a = (14 - month) / 12;
int y = year + 4800 - a;
int m = month + 12 * a - 3;

int jdn = day + (153*m+2)/5 + 365*y + y/4 - y/100 + y/400 - 32045;

return jdn;
}

// 年月日からユリウス通日を返す関数
void inverse_julian_day_number(int jdn, int *year, int *month, int *day){
// 複数の値を返すときは、ポインタで返します

```

```

/*
int n = jdn - 2400001 + 678881;
printf("inv_jdn: n: %d\n", n);
int a = 4*n + 3 + 4*floor(3.0/4.0 * (floor(4*(n+1)/146097) + 1));
int b = 5 * floor((a%1461)/4) + 2;

int y = a / 1461;
int m = b / 153;
int d = (b % 153) / 5;

// y = year + (month - 3) / 12;
// m = (month - 3) % 12;
// d = day - 1;

// であるから、
day = d + 1;
month = (m + 3);
if (month > 12) {
    month = month - 1;
}
*/
// https://en.wikipedia.org/wiki/Julian_day
int f = jdn + 1401 + ((( 4 * jdn + 274277) / 146097) * 3) / 4 - 38;

int e = 4 * f + 3;
int g = (e % 1461) / 4;
int h = 5 * g + 2;
int D = (h % 153) / 5 + 1;
int M = (h / 153 + 2) % 12 + 1;
int Y = (e / 1461) - 4716 + (12 + 2 - M) / 12;

*year = Y;
*month = M;
*day = D;
}

```

## 2.カレンダー。

## calendar.c

```

*****
*
*
* 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つてみましょう。
- \* 「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。
- \* (西暦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ることで、曜日が計算出来ます)
- \*

```
*****
```

```
/
```

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

// ツェラーの公式(Zeller's congruence)により 曜日を返す関数
// 戻り値が 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
int zeller(int year, int month, int day);

// 今月が何日まであるかを返す
int last_day_of_month(int year, int month);

int main(int argc, char const *argv[]) {
    //
    // コマンドラインからの引数がなければ、使い方表示
    //
    if (argc == 1) {
        printf("【使い方】\n");
        printf("カレンダーを表示します。\n");
        printf("2016年 6月のカレンダーであれば\n");
        printf("%s 2016 6\n", argv[0]);
        printf("の様に入力して下さい。\n");
        exit(1); // プログラム終了
    }

    //
    // 変数宣言
    //
    int year;          // 西暦年
    int month;         // 月
    int day;           // 日(カレンダー表示用)

    char *err_str;     // 数値に変換出来なかった文字

    int day_of_week; // 曜日
    int week_number; // 月の第何週か？
    char *days_name[] = { "日", "月", "火", "水", // 曜日の名前
```

```

    "木", "金", "土" };

int last_day; // 今月は何日が最後の日か？

int h; // 今月1日が何曜日であるか？(ツェラーの公式)

// 簡易エラーチェック
//

// 数値変換不可の文字列の長さを調べる。
year = strtol(argv[1], &err_str, 10); // 10進数として変換する
if (strlen(err_str) != 0){
    // エラーメッセージ出力
    printf("%s は、西暦年として認識出来ませんでした。\\n", err_str);
    exit(1); // プログラム終了
}

// 数値変換不可の文字列の長さを調べる。
month = strtol(argv[2], &err_str, 10); // 10進数として変換する
if (strlen(err_str) != 0){
    // エラーメッセージ出力
    printf("%s は、月として認識出来ませんでした。\\n", err_str);
    exit(1); // プログラム終了
}

// カレンダー表示処理
//

// カレンダーの年月と曜日名を表示
printf("\n %4d 年 %2d 月\\n", year, month);
for (day_of_week = 0; day_of_week <= 6; day_of_week++){
    printf(" %s", days_name[day_of_week]);
}
printf("\\n");

// 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。
// 戻り値が 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
h = zeller(year, month, 1);

// 今月の最終日を求める
last_day = last_day_of_month(year, month);

// 水曜日始まりの月の場合、

```

```

// 日、月、火 の欄に日付を表示させたくないでの、
// day = 1 - h からスタートする
day = 1 - h;
day_of_week = 0;
do {
    if (day < 1){
        printf("    "); // 1日以前なら空欄を出力
        day++;
    } else {
        printf(" %2d", day);
        day++;
    }
    day_of_week++;
    if (day_of_week % 7 == 0) {
        printf("\n"); // 週送り
    }
} while (day <= last_day);
printf("\n");

return 0;
}

```

```

// ツェラーの公式(Zeller's congruence)により 曜日を返す関数
// 戻り値が 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
int zeller(int year, int month, int day){
    int y, m, d;
    int h;

    // 年月日の設定
    // 但し1月、2月は前年の13月、14月として計算
    if (month == 1 || month == 2){
        y = year - 1;
        m = month + 12;
        d = day;
    } else {
        y = year;
        m = month;
        d = day;
    }

    // 曜日の算出
    h = (y + y/4 - y/100 + y/400 + (13*m+8)/5 + d) % 7;

```

```
    return h;
}

// 今月が何日まであるかを返す
int last_day_of_month(int year, int month){
    switch(month){
        case 2:
            if ( (year % 4 == 0 && year % 100 != 0) || year % 400 == 0 ){
                return 29;
            } else {
                return 28;
            }
        case 4:
        case 6:
        case 9:
        case 11:
            return 30;

        case 1:
        case 3:
        case 5:
        case 7:
        case 8:
        case 10:
        case 12:
            return 31;
        default:
            return 0;
    }
}
```

## 解答例 ★★★★★★☆

2. ポーカー

poker.c

```
/*=====
/*
 * ポーカー
 */
/* https://ja.wikipedia.org/wiki/ポーカー・ハンドの一覧 */
/* https://simple.wikipedia.org/wiki/Poker#Hands */
=====*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#define BUFFER_SIZE 255 // 文字列用のバッファサイズ

// printfを簡単にするためのマクロ定義
#define prd(dt) printf(#dt ":%d\n", dt)
#define prs(dt) printf(#dt ":%s\n", dt)
#define prx(dt) printf(#dt ":%x\n", dt)

// カードの種類
typedef enum { CLUBS = 4, DIAMONDS, HEARTS, SPADES } suits;
typedef enum {
    NO_PAIR,           // 役無し
    ONE_PAIR,          // ワンペア
    TWO_PAIR,          // ツーペア
    THREE_OF_A_KIND,   // スリーカード
    STRAIGHT,          // ストレート
    FLUSH,             // フラッシュ
    FULL_HOUSE,         // フルハウス
    FOUR_OF_A_KIND,    // フォーカード
    STRAIGHT_FLUSH,    // ストレートフラッシュ
    // HANDS_COUNT,      //
} hands_type;

// プロトタイプ宣言
```

```

void init_card();
void init_player_hands(char player_hands[3][6]);
void init_score();

void disp_hands();
void disp_score();
void disp_score();

void disp_card(char mycard);

int set_card(char mycard,int player);
void calc_card();
char get_card();
void poker_hands(int player);
int my_random(int n);
void pause();
void usage(char const *argv[]);

int is_no_pair(int player);
int is_one_pair(int player,int seach);
int is_two_pair(int player);
int is_three_of_a_kind(int player);
int is_straight(int player);
int is_flush(int player);
int is_full_house(int player);
int is_four_of_a_kind(int player);
int is_suit(int player,int search);

// 文字コードとビット演算の学習を兼ねて、
// 以下のようにカードを表現する
*****
*   A234567890JQK
* C ABCDEFGHIJKLMNOP
* D QRSTUVWXYZ[\]
* H abcdefghijklm
* Sqrstuvwxyz{|}
*
* 例) ABCTd と書くと
* C(クローバー)のA,2,3,
* D(ダイヤ)の4,
* H(ハート)の4でワンペア
*****
// player_hands[1]:EJKLM
// player_hands[2]:ejklm

```

```

// cards[6][17]
//          | A 2 3 4 5 6 7 8 9 0 J Q K A | 1 2
// -----+-----+-----+
// Clubs    | 0 0 0 0 1 0 0 0 0 1 1 1 1 0 | 13 0
// Diamonds | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 0
// Hearts   | 0 0 0 0 2 0 0 0 0 2 2 2 2 0 | 0 13
// Spades   | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 | 0 0
// -----+-----+-----+
// Player 1 | 0 0 0 0 1 0 0 0 0 1 1 1 1 0 | 0 0
// Player 2 | 0 0 0 0 1 0 0 0 0 1 1 1 1 0 | 0 0
// プレーヤー1 の役は フラッシュです
// プレーヤー2 の役は フラッシュです

// グローバル変数
char cards[6][17];           // 1-K + A で14種類 player1/2の合計用に2ます追加
                               // 10,J,Q,K,A のストレート判定をしやすくするため、
                               // K の次に Aを設けている
int score[3][10][3];         // 役の強弱を判定するために用いる score[0] 未使用

int main(int argc, char const *argv[]) {
    int i;
    char ss[BUFFER_SIZE+1];    // ファイルから一行読み込むための作業用
    FILE *fp;                  // ファイルポインタ
    int player;                // 1ならプレーヤー1、2ならプレーヤー2
    char mycard;               // 一枚引いたカード
    char player_hands[2+1][5+1]; // プレーヤーの手元にある5枚のカード
                               // 添字の0は未使用

    // 初期化処理
    init_card();
    init_player_hands(player_hands);
    init_score();

    // コマンドライン引数なしなら、使い方表示して終了
    if (argc < 2){
        usage(argv);
        exit(1);
    }

    // -f ファイルからの読み取りオプションなら
    if (strcmp(argv[1], "-f") == 0){
        if ((fp=fopen(argv[2], "r")) == NULL){
            // エラーメッセージ表示
            printf(" %s を開けませんでした。\\n", argv[2]);
            exit(1);
        }
    }
}

```

```

}

player = 1;
while(1){
    // ファイルから手を読み込む
    fgets(ss, BUFFER_SIZE, fp);
    if (strchr(ss, '*') == NULL && strchr(ss, '/') == NULL){
        // コメント行でなければ、読み取った手をセットする
        strcpy(player_hands[player], ss);
        player_hands[player][5] = '\0';
        if (player == 2){
            break;
        } else {
            player++;
        }
    }
}
fclose(fp);

// 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
for (player = 1; player <= 2; player++){
    for (i = 0; i < 5; i++){
        mycard = player_hands[player][i];
        // 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、
        // エラーチェックはしていない
        set_card(mycard, player);
    }
}

// -m 交互入力オプションなら
} else {
    for (player = 1; player <= 2; player++){
        for (i = 0; i < 5; i++){
            do {
                mycard = get_card(); // ランダムにカードを引く
                // prx(mycard); // 引いたカードを表示(debug)
                disp_card(mycard); // 引いたカードを表示
                pause();
                player_hands[player][i] = mycard;
            } while(set_card(mycard, player) != 0); // 既に引いたカードなら、引き直す
        }
    }
}
}

```

```

// playerの手元にあるカードを表示
prs(player_hands[1]);
prs(player_hands[2]);

// 得点計算
calc_card();

// 役を計算
poker_hands(1); //player1 の役を計算
poker_hands(2); //player2 の役を計算

// カードの表示
disp_hands();

// 得点表示
disp_score();

// 役を表示
for (player = 1; player <= 2; player++){
    printf("プレーヤー%d の役は ", player);
    switch (score[player][9][0]/1000){
        case 8: printf("ストレートフラッシュです\n"); break;
        case 7: printf("フォーカードです\n"); break;
        case 6: printf("フルハウスです\n"); break;
        case 5: printf("フラッシュです\n"); break;
        case 4: printf("ストレートです\n"); break;
        case 3: printf("スリーカードです\n"); break;
        case 2: printf("ツーペアです\n"); break;
        case 1: printf("ワンペアです\n"); break;
        case 0: printf("ぶたです\n"); break;
    }
}

// 勝敗表示
if (score[1][9][0] > score[2][9][0]){
    printf("プレーヤー1の勝ちです\n");
} else if (score[1][9][0] < score[2][9][0]){
    printf("プレーヤー2の勝ちです\n");
} else {
    printf("引き分けです\n");
}
return 0;
}

// カード配列の初期化

```

```

void init_card(){
    int i, j;
    for (j = 0; j < 6; j++){
        for (i = 0; i < 17; i++){
            cards[j][i] = 0;
        }
    }
}

// 得点配列の初期化
void init_score(){
    int i, j, player;
    for (player = 0; player <= 2; player++){
        for (i = 0; i < 10; i++){
            for (j = 0; j < 3; j++){
                score[player][i][j] = 0;
            }
        }
    }
}

// プレーヤーの手にしているカードの初期化
void init_player_hands(char player_hands[3][6]){
    int i, j;
    for (j = 0; j < 3; j++){
        for (i = 0; i < 6; i++){
            player_hands[j][i] = 0;
        }
    }
}

// カードの表示
void disp_hands(){
    int i, j;
    printf("\n");
    printf("          | A 2 3 4 5 6 7 8 9 0 J Q K A |  1  2 \n");
    printf("-----+-----+-----+-----+-----\n");
    for (j = 0; j < 6; j++){
        switch (j){
            case 0: printf("Clubs    | "); break;
            case 1: printf("Diamonds | "); break;
            case 2: printf("Hearts   | "); break;
            case 3: printf("Spades   | "); break;
            case 4: printf("Player 1 | "); break;
            case 5: printf("Player 2 | "); break;
        }
    }
}

```

```

}

for (i = 1; i < 17; i++){
    if (i < 15){
        printf("%1d ", cards[j][i]);
    } else {
        printf("%2d ", cards[j][i]);
    }
    if (i == 14){
        printf("|\n");
    }
}
printf("\n");
if (j == 3){
    printf("-----+-----+-----+-----\n");
}
printf("\n");
}

// 得点の表示
void disp_score(){
    int i;
    printf("          Player1     Player2\n");
    printf("-----+-----+-----\n");
    for (i = 9; i >= 0; i--){
        switch(i){
            case 9:printf("<<MAX>> | ");break;
            case 8:printf("S.Flush | ");break;
            case 7:printf("4 cards | ");break;
            case 6:printf("Full H. | ");break;
            case 5:printf("Flush | ");break;
            case 4:printf("Straight | ");break;
            case 3:printf("3 cards | ");break;
            case 2:printf("2 pair | ");break;
            case 1:printf("1 pair | ");break;
            case 0:printf("no pair | ");break;
        }
        printf("%4d %3d %3d    %4d %3d %3d\n",
               score[1][i][0], score[1][i][1], score[1][i][2],
               score[2][i][0], score[2][i][1], score[2][i][2]);
    }
    printf("\n");
}

// カード管理配列へ、プレーヤーの持っているカードを書き出す

```

```

int set_card(char mycard, int player){
    int suits; // 三つ葉、ダイヤ、ハート、スペード
    int rank; // トランプに書かれている数字

    suits = ((mycard & 0xf0) >> 4) - 4;
    rank = mycard & 0x0f;

    // 例 mycard = 'A' の場合
    // suits = ((mycard & 0xf0) >> 4) - 4;
    // 'A' = 0x41 = 0b0100_0001;
    // 'A' & 0xf0
    // 0x41 & 0xf0
    // 0b0100_0001
    // &) 0b1111_0000
    // -----
    // 0b0100_0000

    // 0b0100_0000 >> 4 // 右へ4ビットシフト
    // 0b0000_0100
    // 0b0000_0100 - 4 = 0; // suits = 0となる

    // mycard & 0x0f;
    // 'A' = 0x41 = 0b0100_0001;
    // 'A' & 0x0f
    // 0x41 & 0x0f
    // 0b0100_0001
    // &) 0b0000_1111
    // -----
    // 0b0000_0001 // 下4ビットを取り出すことが出来た
    // 0b0000_0001 = 1 なので rank(トランプの数字)は1と判明する

    if (cards[suits][rank] == 0){
        cards[suits][rank] = player;
        if (rank == 1){
            cards[suits][14] = player; //エースの時
        }
        return 0; //正常終了
    }
    return cards[suits][rank];
    //どのプレーヤーで用いられているか返す
}

// 役の計算
void calc_card(){
    int i, j;
}

```

```

int sum;
int player;
int mul;
int delta;

// フラッシュ？
for (player = 1; player <= 2; player++){
    for (j = 0; j <= 3; j++){
        sum = 0;
        for (i = 1; i <= 13; i++){
            if (cards[j][i] == player){
                sum += (cards[j][i] / player);
            }
        }
        cards[j][14+player] = sum;
    }
}

for (player = 1; player <= 2; player++){
    for (j = 0; j <= 3; j++){
        if (cards[j][14+player] == 5){
            for (i = 14; i >= 5; i--){
                if (cards[j][i] == player){
                    cards[j][14+player] = i;
                    break;
                }
            }
        }
    }
}

// ペア？
for (player = 1; player <= 2; player++){
    for (j = 1; j <= 14; j++){
        sum = 0;
        for (i = 0; i <= 3; i++){
            if (cards[i][j] == player){
                sum += (cards[i][j]/player);
            }
        }
        cards[3+player][j] = sum;
    }
}

// ストレート？

```

```

for (player = 1; player <= 2; player++){
    for (delta = 0; delta <= 9; delta++){
        mul = 1;
        for (i = 1+delta; i <= 5+delta; i++){
            mul *= cards[3+player][i];
        }
        if (mul != 0){
            cards[3+player][14+player] = --i;
            break;
        }
    }
}
}

// 役の判定
void poker_hands(int player){
    int work; // 作業用変数
    int suit;
    int i;

    work = is_four_of_a_kind(player);
    suit = is_suit(player, work);
    score[player][7][1] = work;
    score[player][7][2] = suit;

    work = is_flush(player);
    suit = work % 10;
    score[player][5][1] = work / 10;
    score[player][5][2] = suit;

    work = is_straight(player);
    suit = is_suit(player, work);
    score[player][4][1] = work;
    score[player][4][2] = suit;

    work = is_three_of_a_kind(player);
    suit = is_suit(player, work);
    score[player][3][1] = work;
    score[player][3][2] = suit;

    work = is_two_pair(player);
    suit = is_suit(player, work);
    score[player][2][1] = work;
    score[player][2][2] = suit;
}

```

```

work = is_one_pair(player,14);
suit = is_suit(player, work);
score[player][1][1] = work;
score[player][1][2] = suit;

work = is_no_pair(player);
suit = is_suit(player, work);
score[player][0][1] = work;
score[player][0][2] = suit;

// フルハウス？
if (score[player][3][1] != 0 && score[player][1][1] != 0){
    score[player][6][1] = score[player][3][1];
    score[player][6][2] = score[player][3][2];
}

// ストレートフラッシュ？
if (score[player][4][1] != 0 &&
    score[player][5][1] != 0){
    score[player][8][1] = score[player][5][1];
    score[player][8][2] = score[player][5][2];
}

for (i = 8; i >= 0; i--){
    score[player][i][0] = score[player][i][1] * 10 + score[player][i][2];
}

for (i = 8; i >= 0; i--){
    if (score[player][i][0] != 0){
        score[player][9][0] = score[player][i][0] + i * 1000;
        score[player][9][1] = score[player][i][1];
        score[player][9][2] = score[player][i][2];
        break;
    }
}
}

// フラッシュか判定
int is_flush(int player){
    int i;
    for (i = 0; i <= 3; i++){
        if (cards[i][14+player] >= 5){
            return cards[i][14+player] * 10 + i + 1;
        }
    }
}

```

```
    return 0; // フラッシュではなかった
}

// ストレートフラッシュか判定
int is_straight(int player){
    return cards[3+player][14+player];
}

// フォーカードか判定
int is_four_of_a_kind(int player){
    int i;
    for (i = 2; i <= 14; i++){
        if (cards[3+player][i] == 4){
            return i;
        }
    }
    return 0;
}

// スリーカードか判定
int is_three_of_a_kind(int player){
    int i;
    for (i = 2; i <= 14; i++){
        if (cards[3+player][i] == 3){
            return i;
        }
    }
    return 0;
}

// ツーペアか判定
int is_two_pair(int player){
    int work;
    if ((work=is_one_pair(player, 14)) == 0){
        return 0;
    }
    if ((is_one_pair(player, work))==0){
        return 0;
    }
    return work;
}

// 10のペアなら10を返す
// ペアが見つからなければ0を返す
int is_one_pair(int player, int search){
```

```

int i;
for (i = search; i >= 2; i--){
    if (cards[3+player][i] == 2){
        return i;
    }
}
return 0;
}

// ノーペアか判定
int is_no_pair(int player){
    int i;
    for (i = 14; i >= 2; i--){
        if (cards[3 + player][i] == 1){
            return i;
        }
    }
    return 0; //error
}

// rank == 13の時、13は何のマークかを返す
int is_suit(int player,int rank){
    int i;
    for (i = 3; i >= 0; i--){
        if (cards[i][rank] == player){
            return i + 1;
        }
    }
    return 0; // rankの数のカードは持っていない
}

// アスキーアートでカードを表示
void disp_card(char mycard){
    int suits;      // 三つ葉、ダイヤ、ハート、スペード
    char rank;       // トランプに書かれている数字

    suits = (mycard & 0xf0) >> 4;
    rank = mycard & 0x0f;
    switch (rank){
        case 1: rank = 'A'; break;
        case 11: rank = 'J'; break;
        case 12: rank = 'Q'; break;
        case 13: rank = 'K'; break;
    }
    // printf("mycard: %x suits: %x, rank: %x\n", mycard, suits, rank);
}

```

```

switch (suits){
    case CLUBS:
        if ( 2 <= rank && rank <= 10){
            printf("      o \n");
            printf("    ( ) \n");
            printf("  / %2d \\ \n", rank);
            printf("c_____o\n");
            printf("      ^     \n");
            printf("      / \_\\   \n");
        } else {
            printf("      o \n");
            printf("    ( ) \n");
            printf("  / %c \\ \n", rank);
            printf("c_____o\n");
            printf("      ^     \n");
            printf("      / \_\\   \n");
        }
        break;
    case DIAMONDS:
        if ( 2 <= rank && rank <= 10){
            printf("      /\\     \n");
            printf("    /   \\   \n");
            printf("  /       \\ \n");
            printf(" \\ %2d   / \n", rank);
            printf(" \\   /   \\ \n");
            printf("   \\ /     \n");
        } else {
            printf("      /\\     \n");
            printf("    /   \\   \n");
            printf("  /       \\ \n");
            printf(" \\ %c   / \n", rank);
            printf(" \\   /   \\ \n");
            printf("   \\ /     \n");
        }
        break;
    case HEARTS:
        if ( 2 <= rank && rank <= 10){
            printf(" /\\ \\ /\\ \\ \n");
            printf(" |     | \n");
            printf(" \\ %2d   / \n", rank);
            printf(" \\   /   \n");
            printf("   \\ /     \n");
            printf("       \n");
        } else {

```

```

        printf(" /\_\_/\_\n");
        printf(" |       | \n");
        printf(" \  %c   /\n", rank);
        printf("   \ /   \n");
        printf("     \/\n");
        printf("           \n");
    }
    break;
case SPADES:
    if ( 2 <= rank && rank <= 10){
        printf("     /\_\n");
        printf("   / \ \n");
        printf(" /     \\\n");
        printf(" |   %2d   | \n", rank);
        printf(" \_\_   _/\n");
        printf("   /_\_\n");
    } else {
        printf("     /\_\n");
        printf("   / \ \n");
        printf(" /     \\\n");
        printf(" |   %c   | \n", rank);
        printf(" \_\_   _/\n");
        printf("   /_\_\n");
    }
    break;
}
}

/* 0-n未満の数を返す */
int my_random(int n){
    return clock() % n;
}

// カードをランダムに引く
char get_card(){
    int suits; // 三つ葉、ダイヤ、ハート、スペード
    int rank; // トランプに書かれている数字

    suits = my_random(4);
    rank = my_random(13) + 1;
    return ((suits+4) << 4) + rank;
}

// 一時停止
void pause(){

```

```

printf(" === press return key === ");
while(getchar() != '\n');
}

// 使い方表示
void usage(char const *argv[]){
    printf(" --- 使い方 --- \n");
    printf("1)");
    printf("%s -f filename\n", argv[0]);
    printf("指定されたファイルからplayer1, player2の手を読み込みます\n");
    printf("\n");
    printf("2)");
    printf("%s -m\n", argv[0]);
    printf("交互にランダムでカードを引きます\n");
    printf("\n");
}

```

### card.txt

```

*****
*   A234567890JQK
*   C ABCDEFGHIJKLMNOP
*   D QRSTUVWXYZ[\]
*   H abcdefghijklm
*   Sqrstuvwxyz{|}
*
* 例) ABCTd と書くと
* C(クローバー)のA, 2, 3,
* D(ダイヤ)の4,
* H(ハート)の4でワンペア
*****
/* プレーヤー1が所有するカード */
EJKLM
/* プレーヤー2が所有するカード */
ejklm

```

# おまけ

自作で入出力関係をいろいろ創ってみました。

my\_io.h

```
#include <stdlib.h>

#define TRUE 1
#define FALSE 0
#define SUCCESS 1
#define FAIL -1

/*****************/
/* 関数名      : get_hex                      */
/* 機能説明    : 8桁の2進数をssに取得する        */
/* 引数        : ss 数字を格納したい配列        */
/*             digit 衡数                      */
/* 返り値      : SUCCESS:数字が入力された時     */
/*             FAIL   :数字以外が入力された時     */
/* 備考        : 動作確認用にprintf文を入れているが、 */
/*             : 実際には、コメントアウトすること */
/*****************/

int get_hex(char ss[], int digit){
    int ch;
    int cnt = 0;
    while( (ch=getchar()) != EOF){
        printf("ch = %d\n",ch);
        if (cnt == 0 && ch == '\n'){
            printf("一文字以上入れてください\n");
            return FAIL;
        }

        //16進数0~9 or a~fでないなら、
        // isxdigitでも良い
        if ( !(( '0' <= ch && ch <= '9') ||
                ('a' <= ch && ch <= 'f')) ){
            if (ch == '\n' || ch == EOF){
                return SUCCESS;
            }
            printf("16進数を入力して ch = %x\n",ch);
            // そうでないならエラー表示
        }
    }
}
```

```

printf("1 6進 cnt = %d\n",cnt);
while((ch = getchar()) != '\n'){
    printf("ch = %x \n",ch);
    ; //読み飛ばし
}
printf("0 / 1 を入力して while 後まで来ました。 \n");

return 2; //エラーコード2を返す
} else if (cnt >= digit){ // 9bit以降読み飛ばし
printf("5桁以降読み飛ばします\n");
printf("cnt = %d digit = %d\n",cnt,digit);
while(getchar() != '\n'){
    ; //読み飛ばし
}
printf("5桁以降読み飛ばしました\n");
return SUCCESS;
} else if (cnt != 0 && ch == '\n'){
printf("正常終了します\n");
return SUCCESS;
} else {
ss[cnt] = ch;
cnt++;
}
}

return SUCCESS;
}

/*****************************************/
/* 関数名      : disp_sary()          */
/* 機能説明   : 文字配列の内容を表示する */
/* 引数        :                      */
/* 返り値      :                      */
/*****************************************/
void disp_sary(char array[], int array_size, char *array_name){
int i;
for (i = 0; i < array_size; i++){
printf("%s[%d] = %x\n", array_name, i, array[i]);
}
}

/*****************************************/
/* 関数名      : disp_iary()          */
/* 機能説明   : int配列の内容を表示する */
/* 引数        :                      */
/* 返り値      :                      */
/*****************************************/

```

```

/*****************************************/
void disp_iary(int array[], int array_size, char *array_name){
    int i;
    for (i = 0; i < array_size; i++){
        printf("%s[%d] = %d\n", array_name, i, array[i]);
    }
}

/*****************************************/
/* 関数名      : init_sary()                      */
/* 機能説明    : 文字配列を初期化する            */
/* 引数        :                                */
/* 返り値      :                                */
/*****************************************/
void init_sary(char array[], int array_size){
    int i;
    for (i = 0; i < array_size; i++){
        array[i] = '\0';
    }
}

/*****************************************/
/* 関数名      : init_iary()                      */
/* 機能説明    : int配列を初期化する            */
/* 引数        :                                */
/* 返り値      :                                */
/*****************************************/
void init_iary(int array[], int array_size){
    int i;
    for (i = 0; i < array_size; i++){
        array[i] = 0;
    }
}

/*****************************************/
/* 関数名      : htod(char ss)                    */
/* 機能説明    : 16進数を10進数に変換する        */
/* 引数        : char ss[] 16進数の文字列        */
/* 返り値      : 10進数                          */
/*****************************************/
int htod(char ss[]){
    int i    = 0;
    int sum = 0;
    while(ss[i] != '\0'){
        if(isalpha(ss[i])){

```

```

        sum = sum * 16 + (ss[i] - 'a' + 10);
        i++;
        // sum = sum * 16 + (ss[i++] - 'a' + 10);
        // と一行に書けるが分かりやすさのために
        // 二行で書いている
    } else {
        // '0'を引くことで、文字から数値へ変換できる
        sum = sum * 16 + (ss[i] - '0');
        i++;
    }
}

return sum;
}

/*****************************************/
/* 関数名      : btod(char ss)          */
/* 機能説明    : 2進数を10進数に変換する */
/* 引数        : char ss[] 2進数の文字列 */
/* 返り値      : 10進数                */
/*****************************************/

int btod(char ss[]){
    int i    = 0;
    int sum = 0;
    while(ss[i] != '\0'){
        sum = sum * 2 + (ss[i] - '0');
        i++;
    }
    return sum;
}

/*****************************************/
/* 関数名      : disp_bin(unsigned char num) */
/* 機能説明    : 10進数を2進数として表示する */
/* 引数        : unsigned char num (0-255)   */
/* 返り値      : void                      */
/*****************************************/

void disp_bin(unsigned char num){
    int i, mask;
    for (i = 7; i >= 0; i--){
        mask = 0x01 << i; // bit & 演算用のマスク
        printf("%d", (num & mask) >> i);
    }
    printf("\n");
}

```

```

/*****************************************/
/* 関数名      : get_str(char ss)          */
/* 機能説明    : キーボードからlen文字の文字を取得、          */
/*                 ssに格納する          */
/* 引数        : char ss[] 入力した文字を格納する配列      */
/*                 : len 何文字分格納するか指定する          */
/* 返り値      : キーボードから入力した文字数          */
/*****************************************/

int get_str(char ss[], int len){
    int i = 0;
    int ch = getchar();
    while(1){
        if (ch == '\n'){
            ss[i] = '\0';
            break; //ループから抜け終了
        }
        if (i >= len){ //10文字以上入ったら? 例)abcdefghijkl\0
            while(getchar() != '\n'); //読み飛ばし
            return FAIL;
        } else {
            // ss[i++] = ch; と一行で書くことも出来る
            ss[i] = ch;
            i++;
            ch = getchar();
        }
    }
    return i;
}

/*****************************************/
/* 関数名      : get_num               */
/* 機能説明    : キーボードから数を取得、整数值として返す。 */
/*                 : 負数対応          */
/* 引数        :          */
/* 返り値      : 戻り値は数値          */
/*****************************************/

int get_num(int *error_flag){
    int num = 0;
    int ch;
    int sign_flag = 1; // 正の数なら1, 負の数なら-1

    while((ch = getchar()) != '\n'){
        if (ch == '-'){
            if (sign_flag == 1){
                sign_flag = -1;
            }
        }
    }
}

```

```

    } else {
        *error_flag = FAIL;
        return FAIL;
    }
} else if ( !( '0' <= ch && ch <= '9')){
    *error_flag = FAIL; //エラーを設定
    return FAIL;
} else {
    num = num *10 + ch - '0';
}
}

*error_flag = SUCCESS;
return sign_flag * num;
}

/*
* 関数名      : get_str_num
* 機能説明   : キーボードから数を取得、文字列として返す。
*              負数対応
* 引数        : なし
* 戻り値      : 戻り値は数値
*/
int get_str_num(char num[]){
    int ch;
    int sign_flag = 1;
    int digit = 0;
    while((ch = getchar())!= '\n'){
        if (ch == '-'){
            if (sign_flag == 1){
                sign_flag = -1;
                num[digit] = '-';
                digit++;
            } else {
                return FAIL;
            }
        } else if ( !( '0' <= ch && ch <= '9')) {
            return FAIL;
        } else {
            num[digit] = ch;
            digit++;
        }
    }
    num[digit] = '\0';
    return SUCCESS;
}

```

```

}

/*****************************************/
/* 関数名      : get_float                      */
/* 機能説明    : キーボードから浮動小数点数を取得、          */
/*                 実数値として返す。負数対応                  */
/* 引数        : なし                                */
/* 返り値      : 戻り値は数値                      */
/*****************************************/

int get_float(void){
    float number          = 0.0; // 読み込んだ実数
    char ch;                // 一文字読み込み用
    int sign_flag         = 1; // 負数なら -1;
    int floating_point_flag = 0; // 小数点があらわれたら 1
    int numeric_flag       = 0; // 数になったら 1
    int fraction_part     = 1; // 小数点以下の桁数

    ch = getchar();
    if (ch == '\n'){
        return FAIL;
    } else if (ch == '-'){ // 負数なら
        sign_flag = -1;
    } else if (ch == '.'){
        floating_point_flag = 1;
    } else if (isdigit(ch)){
        fraction_part = ch - '0';
        numeric_flag = 1; // 数が入力されたのでフラグを立てる
    } else {
        numeric_flag = 0;
    }

    while ((ch = getchar()) != '\n'){
        if (isdigit(ch)){
            numeric_flag = 1;
            if (floating_point_flag != 1){
                fraction_part = fraction_part * 10 + (ch - '0');
            } else if (floating_point_flag == 1){
                fraction_part
                    = fraction_part + (ch - '0') * pow(10, -fraction_part);
                fraction_part++;
            }
        } else if (ch == '.'){
            if (floating_point_flag != 1){
                floating_point_flag = 1; // 小数点フラグを立てる
                fraction_part = 1;
            }
        }
    }
}

```

```

        } else {
            return FAIL;
        }
    } else {
        return FAIL;
    }
}

return sign_flag * fraction_part;
}

/*****************************************/
/* 関数名      : get_line                  */
/* 機能説明   : キーボードから一行入力し、ssに格納する      */
/* 引数        : ss 文字列格納用配列          */
/*             string_length 何文字入力するのか          */
/* 返り値      : なし                      */
/*****************************************/
void get_line(char ss[],int string_length){
    int i = 0;
    char ch;

    while((ch = getchar()) != '\n'){
        ss[i] = ch;
        if (i == string_length){
            while(getchar() == '\n'); // 読み飛ばす
        } else {
            i++;
        }
    }
    ss[i] = '\0'; //文字列の最後は'\0'
}

/*****************************************/
/* 関数名      : get_day()                 */
/* 機能説明   : キーボードから入力された文字が      */
/*             YYYYMMDD形式であるか判定する。          */
/* 引数        : なし                      */
/* 返り値      : 日付形式ならばYYYYMMDD形式でlong型で返す。 */
/*             エラーなら-1                  */
/*****************************************/
long get_day(void){
    long date;
    int year  = 0;
    int month = 0;
}

```

```

int day = 0;
int last_day[12+1]
    = { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
int ch;
int i = 0; //カウンタ

//入力
while((ch = getchar()) != '\n'){
    if (!isdigit(ch) || i >= 8){
        while(getchar() != '\n'); //読み飛ばし
        return FAIL; //エラー値を返す
    }

    if (i < 4){
        year = year * 10 + ch - '0';
        i++;
    } else if (i < 6){
        month = month * 10 + ch - '0';
        i++;
    } else if (i < 8){
        day = day * 10 + ch - '0';
        i++;
    }
}

//条件判定
if (!(1 <= month && month <= 12)){
    return FAIL;
}
if ((year % 4 == 0 && year % 100 != 100) || year % 400 == 0){
    last_day[2] = 29; // 閏年の2月は29日まである
}
if (!(1 <= day && day <= last_day[month])){
    return FAIL;
}

return (long)year*10000 + (long)month*100 + (long)day;
}

```